PROVISIONAL APPLICATION UNDER 37 C.F.R. §1.53(c)

TRANSMITTAL FORM

Docket Number: ***TI-30639PS***

Assistant Commissioner For Patents

Washington, D.C. 20231

Dear Sir:

EXPRESS MAIL" Express Mailing Label Number ***EL360245732US***, Date of Deposit: ***February 18, 2000***. I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner For Patents, Washington, DC 20231.

ELIZABETH AUSTIN

Enclosed application parts are:

_	Spec w/Claims	Number of Pages ___
X	Spec w/o Claims	Number of Pages 166
_	Formal drawings	Number of Sheets ___
_	Informal drawings	Number of Sheets ___
_	Other:	

Inventors:

LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY & STATE OR CITY & FOREIGN COUNTRY)
Bertrand	*Pierre*		*Nice, France*
Biscondi	*Eric*		*Nice, France*
Brown	*Katherine*	*G.*	*Coppell, Texas*
Honore	*Frank*	*A.*	*Dallas, Texas*
Sriram	*Sundararajan*		*Dallas, Texas*

TITLE OF INVENTION: ***CORRELATOR COPROCESSOR (UPLINK)***

CORRESPONDENCE ADDRESS: **Ronald O. Neerings**
Texas Instruments Incorporated
P. O. Box 655474, M/S 3999
Dallas, Texas 75265

PHONE: (972) 917-5299
FAX: (972) 917-4418/4417

Was this invention made under a Government contract? X No ___ Yes

Identify contract and the Government agency: _____

Please charge $150 to the deposit account of Texas Instruments Incorporated, Account No. 20-0668.
An original and two copies are enclosed.

Respectfully submitted,

Ronald O. Neerings

Ronald O. Neerings
Reg. No. 34,227

2/18/00
Date

PROVISIONAL APPLICATION ONLY

30639
1379
RON

Correlator Coprocessor (Uplink)

Functional Specification

Preliminary
Ver 0.1

File: \\DNCS39\WISAP\Specs\CCP\CCP_uplink.doc

Department: Application Specific Product / Wireless Communications System

Contributors

Authors
Pierre Bertrand
Eric Biscondi
Kathy Brown
Frank Honore
Sriram Sundararajan

History

Version	Date	Author	Notes
Ver: 0.0	01/05/99	Eric B. / Pierre B.	1
Ver: 0.1	01/21/99	Eric B. / Pierre B.	2

NOTES :

1. Very first draft
2. First Preliminary release.

SUMMARY

TABLE OF FIGURES

1. Introduction

The correlator coprocessor (CCP) is a programmable, highly flexible, vector-based correlation machine that performs CDMA base-station RAKE receiver operations for multiple channels. Because most RAKE receiver functions involve correlations and accumulations, regardless of the particular wireless protocol, a centralized correlation machine can be used for various RAKE receiver tasks like finger despreading and search.

In addition to performing correlations (complex valued), which consist of despreading and coherent accumulation, the correlator coprocessor also accumulates "symbol" energy values (called non-coherent accumulations). For example, it accumulates the early, on-time, and late samples of a RAKE finger; these measurements are used for the finger's code-tracking loop (typically a delay-lock loop or DLL). For search operations, the correlator coprocessor returns the accumulated energy values for a specified window of offsets.

The CCP does not perform "symbol"-rate receiver operations such as channel estimation, maximal-ratio combining (MRC), and de-interleaving, nor feedback loops such as AGC, AFC, and DLL. (For DLL, the CCP supplies the energy values to the feedback loop, but it does not operate on the loop itself.) All these symbol operations are performed on a TMS320C641x DSP.

1.1 Implementation Parameters

Many versions of the CCP may be implemented using this specification. For this reason, some parameters will be referred to with names rather than numbers (e.g. SYS_CLK, MAX_CYCLES). The paragraph in the implementation section, which refers to a particular version, will define these parameters.

- X = 1.2288 MHz, defines the chip rate.
- SYS_CLK = 64X = 78.6432 MHz, defines the system clock.
- MAX_USERS = 64, maximum number of users/channels processed simultaneously by the CCP. In the CCP, it defines the range of the USER_ID index (*see section Tasks, Finger Task instruction set*) and the number of Walsh Blocks in the Walsh Table (*see section Software Interface / Walsh Table*)
- MEAN_FINGERS_PER_USER = 4, mean number of fingers per user/channel: used to define the maximum number of fingers simultaneously processed by the CCP (*see below*).
- MAX_FINGERS = MAX_USERS*MEAN_FINGERS_PER_USER = 256, maximum number of finger tasks simultaneously processed by the CCP. In the CCP, it defines the range of Finger Tasks, i.e. the Task Buffer size for the Rake Datapath (*see section xxx*)
- PS_SIZE = 384, defines the number of chips per Pilot Symbol and per Power Control Bit (312.5 µs).
- SLOT_SIZE = 4 Pilot Symbols, i.e. one Power Control Group (1.25ms).

- FRAME_SIZE = 16 slots, i.e. 16 PCG (20 ms).
- MAX_CYCLES = 16*SYS_CLK / X = 1024, maximum number of Rake correlation cycles per CCP iteration (16 chips).
- PAR_SEARCH = 2, parallelism level implemented on a search process (independently of the on-time and late-time simultaneous processing).
- MIN_WIN_SIZE = 16*PAR_SEARCH = 32 chips, search window size granularity.
- MAX_WIN_SIZE = PAR_SEARCH*MAX_CYCLE = 2048 chips, maximum search window size for a DPE/Search task.
- MAX_DPE = PAR_SEARCH*MAX_CYCLE/MIN_WIN_SIZE = 64, maximum number of simultaneous DPE/Search tasks.
- MAX_FINGERS_PER_USERID = 8, maximum number of fingers allocated to the same User_ID (*Walsh Block*)

1.2 Wireless Protocol

This version of the correlator coprocessor supports IS95 and cdma2000 1xRTT receive operations. (1.2288 MHz chip rate). Future revisions will support cdma2000 3xRTT and 3GPP (3.84 MHz chip rate).

1.3 System Requirements

1.3.1 Input clock

Let 1X be a clock rate of 1.2288MHz, i.e. the baseline IS95/1xRTT chip rate. The correlator coprocessor requires the following input clocks:

- System clock: 64X clock (SYS_CLK), 78.6432MHz.
- Receive Sample clock: 8X clock or 4X clock.

These two clocks must be synchronous and phase aligned.

Using a 4X clock reduces CCP input buffer requirements and A/D power at the cost of reduced signal-to-noise ratio (SNR). Receive sample clock only affects the Input Buffer and not the core of the correlator coprocessor.

1.3.2 Receiver I/Q Samples

In-phase (I) and quadrature (Q) samples are up to 6 bits each. The CCP receives multiple I/Q samples from multiple antennas. The buffers that store the I/Q samples are outside of the core of correlator coprocessor. Up to 18 antenna sources may be supported. Each source may be sampled at 4 or 8 times the chip rate.

1.3.3 System Bus

The TMS320C641x DSP accesses to all CCP's buffers through the External Memory Interface (EMIF_B). This interface is a 16-bit data interface providing a maximum

bandwidth of 266 Mbytes/s. Enhanced DMA controller (available on the DSP) performs all data transfers. The CCP also provides a Finger Symbol Buffer (FSB) external i/f that allows the another block (for example a potential MRC sub-block) to access the Finger Symbols directly in the FSB without any DSP intervention.

1.4 Features

The correlator coprocessor performs correlations and accumulations (coherent and non-coherent). The basic operation is the correlation, commonly referred to as de-spreading (followed by sum-and-dump), which is used to remove the effects of long-code scrambling and Walsh spreading. The correlator coprocessor controls correlations and accumulations in certain ways to achieve all RAKE and searcher operations. The correlator is a multi-tasking machine. For example, a finger operation is called a Finger task.

The tasks are:

- 1xRTT Finger task
- IS-95 Finger task
- 1xRTT DPE/Search task
- IS-95 DPE/Search task

These tasks and their programmable features are briefly described in the sections that follow. Exact details on all tasks are found in Section 7.

1.4.1 Finger Task

In a finger task, de-spreading is performed, resulting in complex symbols (I&Q) that are output. Each finger operation can accommodate multiple Walsh Code channels, all with a common set of parameters defined in the Finger Task Instruction and each channel may then have specific parameters that are defined in the Walsh sub-task instructions. Each Walsh sub-task has for example a specific spreading factor that may be set from 4 up to 512.

Each finger operation supports one set of early/on-time/late (EOL) energy measurements that can be processed on the pilot channel or on any other data channel identified by its Walsh code.

In the former case, the length of the coherent accumulation can be specified between 48 and 384 chips in mode0 and between 1536 and 24576 chips in model (Power Control Bits are automatically skipped). Then the coherent packets are non-coherently accumulated over N_{NTS} packets. N_{NTS} is a parameter that the user can set from 1 up to 1024.

In the latter case, the length of the coherent accumulation is the symbol length in chips, given by the SF. Then, the coherent symbols are non-coherently accumulated over N_{NTS} symbols. N_{NTS} is the same parameter as above that can be set from 1 up to 1024.

In addition, a finger task also performs coherent accumulations on the pilot channel to get the raw pilot symbols that can then be used by the DSP to perform the channel estimation. User can choose to perform the coherent accumulation for the pilot channel on PS_SIZE/N_{CE} chips, where N_{CE} is one of the task parameters. That means that the length of the coherent accumulation can vary from 48 up to 384 chips.

Finger tasks run continuously until they are disabled. Despread data symbols (I&Q) are stored in the Finger Symbol Buffer (FSB). Despread pilot symbols, i.e. pilot symbols and power control symbols (I&Q) are written in the Raw Pilot Buffer (RPB) and EOL energy measurements are output in the EOL/DPE Buffer.

1.4.2 Delay Profile Estimation (DPE)/Search Task

This task is used to identify potential multi-paths in a window of offsets and is also used to perform acquisition of new mobile station. (R-ACH Access channel acquisition).
An energy value is returned for every chip or ½ chip offset in the specified window. The DPE/Search task measures energy exactly like the EOL measurement of the Finger task and has exactly the same level of flexibility than the EOL measurement. (See Finger Task description above)
DPE/Search tasks require more than a CCP iteration to complete and stop when they are completed. An interruption may be generated to notify the DSP that the task is completed.

1.5 CCP Resources Allocation

In this section, we briefly describe the different ways that CCP resources can be divided to perform the RAKE receiver functions described in the previous sections.
There are a total of MAX_CYCLES "correlation cycles", or cycles, supported by the CCP. In other words, the CCP has a processing capacity equivalent to MAX_CYCLES discrete Correlators for RAKE fingers and PAR_SEARCH*MAX_CYCLES equivalent Correlators for DPE search. The number of correlation cycles expended for each CCP task is as follows:
• Finger task. Each Walsh channel uses one correlation cycle, independent of its Spreading Factor. Each set of EOL measurements uses three correlation cycles.
• DPE/Search task. Window sizes are a multiple of 16 chips. Every PAR_SEARCH chips of window size uses only one cycle, even though PAR_SEARCH equivalent correlations are performed, or 2* PAR_SEARCH equivalent correlations performed

for the on-time samples and (on-time + ½-chip) samples when the ½ chip option is enabled[1].

The CCP supports MAX_FINGERS simultaneous Finger tasks. Each Finger task support up to 8 Walsh channels. Each Finger task may support 1 set of EOL measurements, but a Finger task making EOL measurements will reduce by one the number of Walsh channels supported on that finger. Hence, Finger tasks may specify up to a maximum of 7 x MAX_FINGERS + 3 x MAX_FINGERS cycles (restricted by the fact that only MAX_CYCLES cycles could be run at any one time).

The CCP supports up to MAX_DPE DPE Search tasks. This number can be supported only if the sum of the sizes of the search windows is less than PAR_SEARCH * MAX_CYCLES. The CCP supports a total of MAX_TASKS tasks in any combination that conforms to the rules above.
All finger tasks together cannot exceed MAX_CYCLES cycles.
All DPE/Search tasks together cannot exceed PAR_SEARCH*MAX_CYCLES cycles.
A disabled task does not consume any cycle.
An enabled task that is not running (*e.g. waiting for a specified slot to start)* consumes one cycle per task.
An enabled task which is not implemented (*e.g. Walsh sub-task disable)* consumes one cycle per task.

1.6 Digital Base Band System Partitioning

The CCP performs all chip-rate processing and energy accumulation according to the tasks that DSP writes to the CCP's Task Buffers to control all CCP's operations.
The TMS320C641x DSP performs all symbol-rate receiver operations such as channel estimation (*including phase and frequency estimation*), de-interleaving, feedback loops such as Automatic Gain Control, and Delay Locked Loop. (For AGC and DLL, the CCP supplies the energy values to the feedback loop, but it does not operate on the loop itself).
The maximal-ratio combining (MRC) may be either implemented in the DSP or in another sub-block controlled by the DSP.

Figure 1-1 shows an example of implementation using the CCP and shows how the CCP could be interfaced to the other component of the receive chain of a Digital Base Band (DBB) hardware configuration. The CCP is responsible for:

- performing the despreading to provide data symbols per finger to the entity in charge of the MRC processing (may be directly either directly the DSP or a another ASIC sub-block),
- performing EOL energy measurements for DLL,

[1] The Search Datapath implements 2*PAR_SEARCH parallel correlation machines.

- performing on-chip and ½-chip correlations and energy measurements for DPE and Search purposes,
- providing raw pilot symbols per finger to the DSP.

DSP uses the computed raw pilot symbols to perform the channel estimation of each finger. Coefficients of the channel estimation are then sent to the entity in charge of the MRC processing. In that particular example, MRC processing is done in hardware, but it could also be processed directly by the DSP.

Using those computed coefficients, the MRC multiplies despread symbols with the channel estimation coefficients and then sums the symbols coming from various fingers (paths) together to provide combined symbols in the Combined Symbol Buffer (CSB).

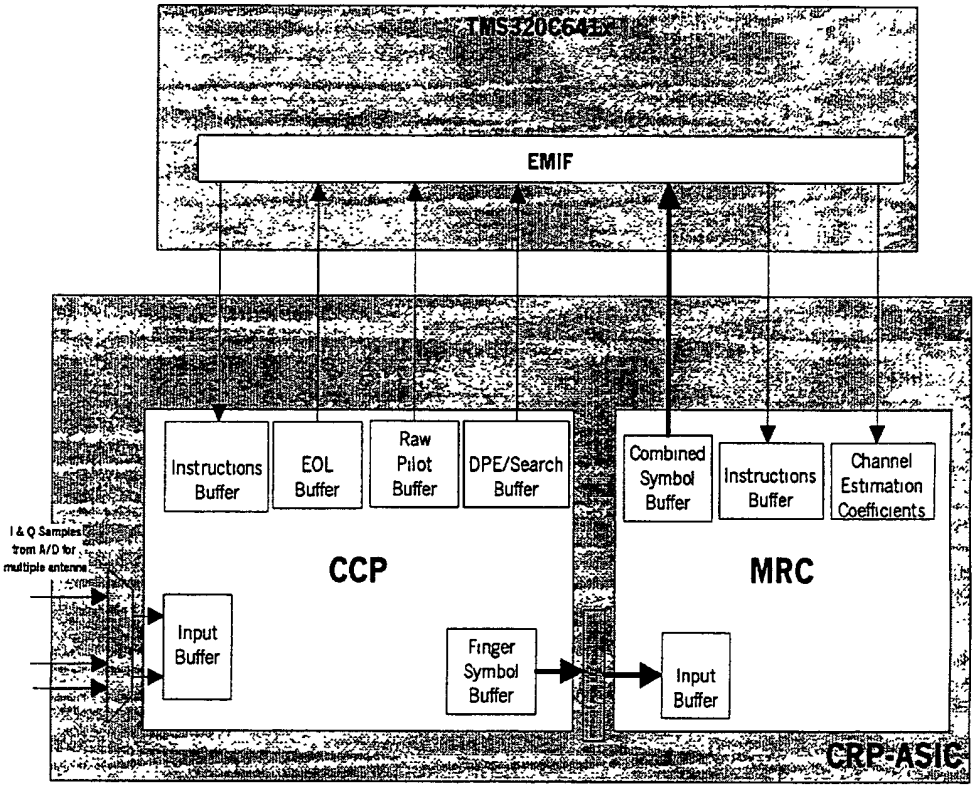


Figure 1-1 : Example of implementation using the CCP

1.7 Roadmap

<TBD>

2. Architecture Overview

2.1 Input Buffer

The Input Buffers store a stream of received I/Q sub-chip samples for processing by the CCP Search or Rake Datapath. Chips are over-sampled by a factor 4X or 8X. These sub-chip samples may come directly from an analog front-end (AFE), digital front-end filtering, or digital interpolation filtering. For each correlation cycle, the CCP selects a set of 16 input chips corresponding to a particular sub-chip sample position (*only 1 sample/chip is fetched*). The CCP may receive input data from multiple sources, for example, to support multiple antennas. Up to 18 input sources are supported. The Input Buffers may be implemented as custom register files.

2.2 Rake Data Path

The Rake datapath is shown on Figure 2-1. It consists of:

- A sample-select function: selects the sample position to be used for all chips of a finger task,
- "multipliers" to multiply chunks of 16 I&Q samples from the input buffer with 16 samples of the complex PN spreading performed in IS-95/1xRTT: (Long Code+Mask) + Short Code + Walsh code (1xRTT only),
- adder trees to generate partial correlations results over 16 chips,
- coherent accumulator (on SF) to perform coherent accumulation on symbols length,
- coherent accumulator (on PS_SIZE/N_{CE}) to perform coherent accumulation over a fraction of pilot symbol length,
- coherent accumulator (on PS_SIZE/N_{CA} or on $SLOT\_SIZE*N_{CA}$), followed by an energy estimator and a non coherent accumulator to perform the early, on-time and late energy measurements. (Energy values are output every frame).

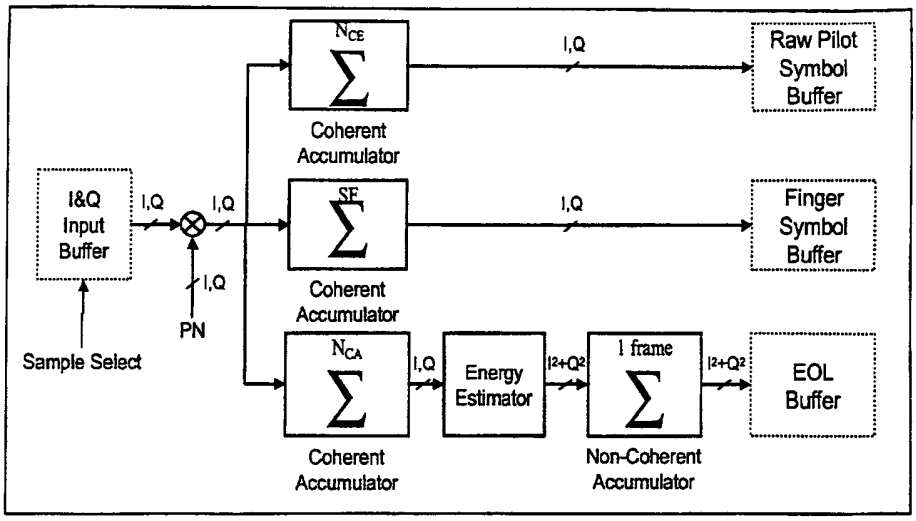


Figure 2-1: Rake datapath

2.3 Search Data Path

The search datapath is shown on Figure 2-2. It consists of:

- "multipliers" to multiply chunks of 16 I&Q samples from the input buffer with 16 samples of complex PN (Long Code + Short Code + Walsh)
- adder trees to generate partial correlations results over 16 chips
- coherent accumulator (on PS_SIZE/N_{CA} or on SLOT_SIZE*N_{CA}), followed by an energy estimator and a non coherent accumulator (on N_{NTS}) to perform energy measurements on each code offset hypothesis and for the on-time and half-chip-delayed sample hypothesis.

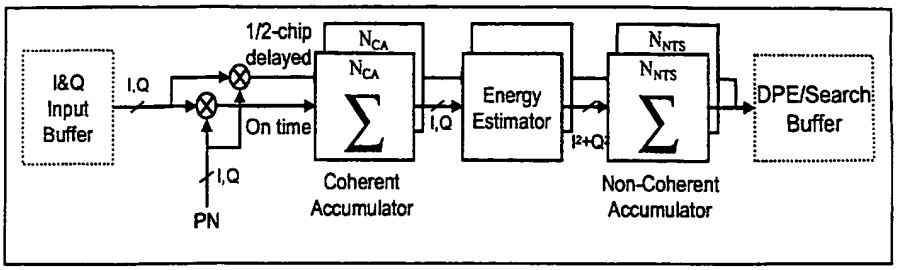


Figure 2-2: Search/DPE datapath

2.4 Datapath precision

The CCP accumulates bits and discards bits at different stages of the datapath. The input from the A/D is 6 bits, and after the adder trees, there are 17 bits. At this point, some bits are discarded. Before writing symbols to the Finger Symbol Buffer, 9 MSB's are discarded, with saturation, for SF = 4; and 1 MSB is discarded, with saturation, for other SF's.

For the symbols passed into the remainder of the datapath (energy estimator), 4 MSB's and 2 LSB's are discarded, with saturation.(*TBC*).

After Coherent Accumulation, there are 22 bits. Of these 22 bits, 18 bits are kept starting from $(13 + \max(5, \log2(Ns)))^{th}$ LSB, where Ns is the number of symbols of coherent accumulation (*TBC*).

After Non-Coherent Accumulation, there are 32 bits. Of these 32 bits, 24 bits are kept starting from $(13 + \max(5, \log2(N_{NS})))^{th}$ LSB,.where N_{NS} is the number of non-coherent accumulations (*TBC*).

2.5 Finger Symbol Buffer

Finger Tasks store I&Q "symbols" in the Finger Symbol Buffer (FSB). This buffer is implemented as a multi-slot (4 slots of (SLOT_SIZE*PS_SIZE) chips) buffer for each Walsh channel.

The Finger Symbol Buffer serves as intermediate storage for downstream symbol-rate processing. Its size is a compromise between area and the rate at which data must be moved to where downstream processing takes place. It is not accessible directly from the DSP's EMIF bus[2]. An FSB External Bus may be defined when MRC processing takes place in hardware, to connect the CCP to the MRC sub-block.

2.6 Raw Pilot Buffer

The Raw Pilot Buffer (RPB) stores the raw pilot symbols of the fingers currently running with a DLL task enabled on the pilot channel[3]. For such fingers, the DSP uses

[2] For debug and test purpose, it might be accessed by the DSP in "*Test Mode*" (TBD)

[3] When the DLL task is not specified on the pilot but on a data channel, no data is output in the Raw Pilot Buffer.

their pilot symbols to perform each finger's channel estimation (carrier's amplitude, phase and frequency).
When a finger task is running, a new raw pilot symbol is written to this buffer every PS_SIZE/N_{CE} chips. The DSP can use the Finger task End-of-slot interrupt event to get new raw pilot symbols.
Similarly to the FSB, the RPB is implemented as a four-radio-slot long circular buffer.
Note that this output provides the pilot unmodulated bits (whole or in fractions) as well as the Power Control Bits. It "saves" then a Finger/Walsh specification dedicated to the Power Control Bits despreading and combining. Moreover, it allows the DSP to demodulate the PCB faster[4].

The size, in bits, of the RPB is:

$$MAX\_FINGERS*N_{CE\_MAX}*2_{(I\&Q)}*16_{(bits)}*4_{(symbols)}*4_{(slots)}$$

2.7 EOL Buffer

The EOL Buffer stores finger EOL measurement results. DSP can access directly this buffer through the EMIF at all times. The EOL Buffer is single-buffered, and new results over-write old ones. When new results are ready, they can be read by the DSP through the EMIF. The Finger task can issue slot-based interrupt event that can be used to signal the availability of new EOL data.

EOL calculation is a continuous process: integration is performed coherently over (PS_SIZE/N_{CA}) chips in mode 0 and over $(SLOT\_SIZE*PS\_SIZE*N_{CA})$ chips in mode 1. These partial results are then incoherently dumped. The non-coherent accumulation is reset on every frame and the energy results are then output once per frame.

2.8 DPE/Search Buffer

The DPE/Search buffer stores DPE and search results. They are directly readable via the EMIF Bus at all times.
Unlike the finger task, which is continuous, the DPE/Search task is one shot. Consequently, the DPE/Search Buffer is single-buffered, and new results over-write old ones. When new results are ready, they may be read on the EMIF Bus directly. When a DPE/Search task finishes, an interrupt may be generated.
DPE energy are output in this buffer after integrating over $N_{NTS}*(PS\_SIZE/N_{CA})$ chips in mode 0 or $N_{NTS}*(SLOT\_SIZE*PS\_SIZE*N_{CA})$ chips in mode1.

[4] Though the PC loop is slower than in 3GPP, we may need to control separately the PC bits and the data: the channel estimation algorithm may take a long time to run and we can't afford its completion + the hardware MRC delay to demodulate the Power Control Bits. Hence the PCB from different fingers will be combined by the DSP using temporary rough coefficients.

2.9 LC & SC Generators

A CCP task specifies the Long Code (LC) and the Short Code (SC) to be generated as well as the code offset. The LS/SC code generators then generate a block of the specified LC/SC codes starting from the specified code offset.
Both "block" and "serial" Gold code generation methods are employed to minimize power dissipation.

2.10 Controller

The control part of the CCP is responsible for actually implementing each of the CCP tasks, and generating appropriate control signals for the datapath. Different flavors of correlations can be done by the datapath by varying the control sequence. When no task is running, downstream control and datapath pipeline stages are gated off to conserve power.

2.11 Global Chip Counters (GCC1 & GCC2)

Global Chip Counters maintain the local timing reference of the CCP. GCCs count the incoming chip samples as they are written into the Input Buffers. GCC1 counts modulo the length of the Long Code (2^{42}-1) and GCC2 counts modulo three times[5] the length of the Short Code (2^{15}). All timing in the CCP is relative to the GCC2, including offsets used in RAKE receiver operations.

2.12 Configuration Tables

The CCP uses a number of configuration tables to specify how it executes each of its tasks. Some tables are used globally, while others are associated with certain tasks. For example, one configuration table contains the Walsh codes associated with a particular Finger task. Configurations come directly from the DSP. All configuration tables are described in detail in Section xx.

2.13 Interrupt Generator

There are three types of interrupts in the CCP. They are task-based interrupts, system interrupts, and error interrupts.

Each CCP task can generate one interrupt. For example, when a DPE task finishes, it may generate an interrupt. Task-based interrupts are mainly used by the host processor for data retrieval, but may be used for other SW/HW synchronization purposes.

Task-based interrupts place status information in one of the four interrupts FIFOs. Each interrupt FIFO is tied to one of the four interrupt lines coming from the CCP.

System interrupts indicate global CCP events: The Task-Update interrupt signals the host processor that task updates are completed.

[5] GCC2 = 3xSC = 4xframe = 98304 chips. This simplifies the internal frame/code hardware control.

An error interrupt is generated whenever an error condition is detected.

2.14 Task Buffers

There are two tasks buffers: one for the Rake Datapath listing 1xRTT/IS-95 Finger Tasks, and one for the Search Datapath, listing 1xRTT/IS-95 DPE tasks. Each Task Buffer contains a list of tasks that the CCP executes. The Task Buffer is read directly by the CCP in order to determine the CCP's current tasks. The Task Buffer is a ping/pong buffer with an individual control for the ping/pong status of each entry in the Task Buffer. The swapping from ping to pong or vice-versa occurs on a Task-Update boundary. The Task-Update interrupt tells the host processor when the transfer completes, and that the updated status bits are available for each task. This mechanism allows synchronization between the host processor and the CCP, which prevents incomplete tasks being read by the CCP.
The CCP supports up to MAX_FINGERS tasks in the Rake Task Buffer and MAX_DPE tasks in the Search Task Buffer.

3. Synchronization to System Time

3.1 Introduction

The internal clock of the BS increments two chip counters GCC1 and GCC2. GCC1 and GCC2 are incremented at chip rate and give the current local position in the long code (42 bits) and in the short code (15 bits)[6] respectively. They provide an absolute Internal Time Base with the accuracy of a chip period as the realignment of their initial states occur more then 37 centuries apart.
An external GPS source supplies time synchronization between this Internal Time Base and Global System Time.
At power-up, the BS performs the measurements from a GPS source of the offsets Δ_1 and Δ_2 of GCC1 and GCC2 with respect to the Long Code and the Short Code in System Time. GCC1+Δ_1 and GCC2+Δ_2 define then the Local Time Base which is synchronized to the CDMA System Time with the accuracy defined by the standard specification. If the GPS is not present, the Local Time Base runs on its own, from the last available GPS information.
Since GCC1 and GCC2 are driven synchronously by the internal clock, they have identical time drift behavior with respect to the GPS. The corrections being the same, the tracking statements use only one of them, say GCC2.
The PN generators of the transmit side and of the receive side share the same GCC1 and GCC2 registers.

[6] Though the short code is 15 bits wide, GCC2 is 17 bits so as to get an integer number of 20 ms frames.

3.2 GPS acquisition

The GPS receiver provides 1PPS pulses (Δt = 1s) phased-aligned on the System Time seconds front. The absolute time of each pulse arrives simultaneously by a periodic "absolute time" message (*along with the location of the BS*) through a serial port on the DSP. The format of the message varies from a manufacturer to another, and in some cases can be programmable: the time to the origin, January 6, 1980, can be displayed in number of seconds or number of seconds in the current day + number of days, etc. The pulses and the messages are synchronous at the output of the GPS receiver but need to be processed by the BS with a real time constraint just tight enough to prevent from any ambiguity between them: $\pm\Delta t/2$.

The pulses trigger the transfer to the DSP memory of the current values of GCC1 and GCC2. In the following we focus on GCC1 (*GCC2 acquisition is identical*).Figure 3-1 and Figure 3-2 illustrates the GPS pulse-based acquisition principle. Figure 3-3 describes the dedicated hardware implementation included in the CCP. The GPS pulse, registered by the CCP internal clock, triggers the transfer of GCC1 into an output register, which is then read by the DSP through the EMIF. The simultaneous timing messages are received by the DSP through a serial port. The DSP receives an interrupt from the GPS receiver to let him know a new message has arrived. The DSP uses this information to store in its memory the GCC1 value from the output register.

Each couple of values [GCC1(i), time message(i)] associated to a GPS pulse front provides an estimation $\hat{\Delta}_1(i)$ of the time (*or chip*) offset between the Long Code in the System Time and in the CCP Internal Time (*GCC1*):

The value $L_{ST}(i)$ of the System Time Long Code on the GPS pulse front is derived from the date i of arrival of the GPS pulse given by the GPS time message: if i is the number of seconds to the origin of the System Time (*January 6, 1980*), then:

$$\hat{L}_{ST}(i) = i \times F_{chip} - \left(2^{42}-1\right) \times round\left[\frac{i \times F_{chip}}{2^{42}-1}\right] \quad ; \quad F_{chip} = chip\ rate$$

$\hat{L}_{ST}(i)$ is a real value and its uncertainty is given by the pulses position accuracy. This noise source is described in [4].

The value $L_{CCP}(i)$ of the CCP Long Code on the GPS pulse front is derived from the associated stored value of GCC1 using the non-biased estimator (*see* Figure 3-1):

$$\hat{L}_{CCP}(i) = \left(GCC1(i) + 0.5\right)$$

$\hat{L}_{CCP}(i)$ is an estimation based on GCC1 samples. The noise induced by this sampling process is evaluated in [4].

$\hat{\Delta}_1(i)$ is given by:

$$\hat{\Delta}_1(i) = \hat{L}_{ST}(i) - \hat{L}_{CCP}(i)$$

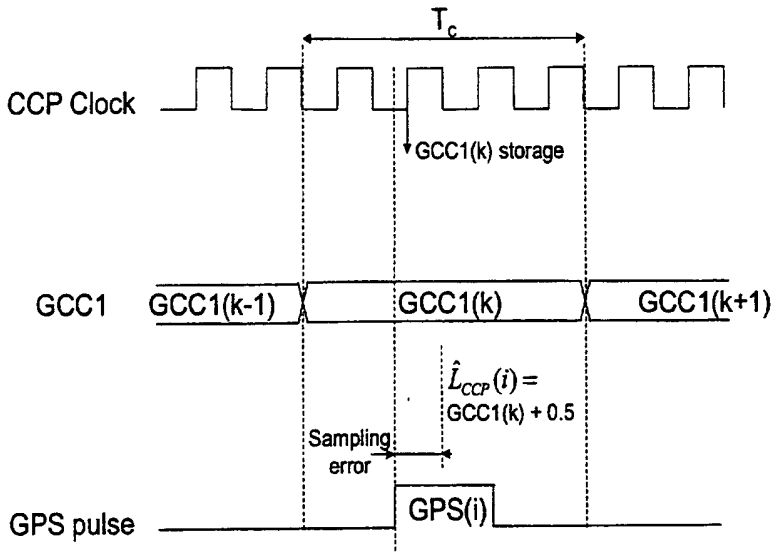


Figure 3-1: Acquisition process

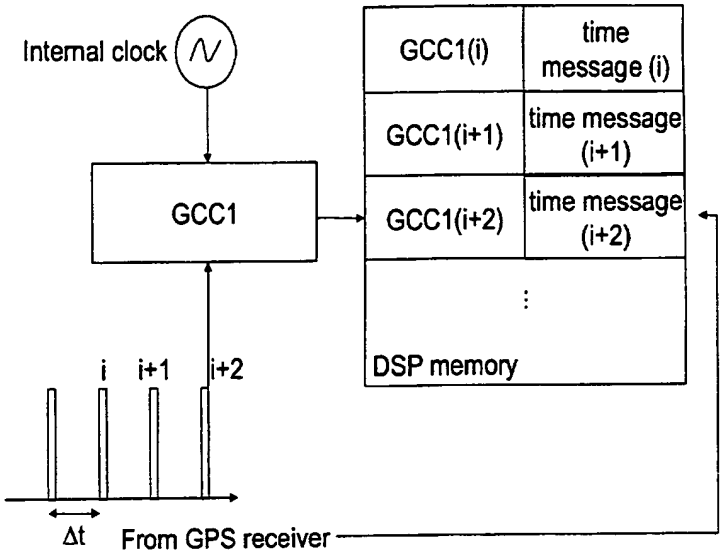


Figure 3-2: GPS pulses acquisition principle

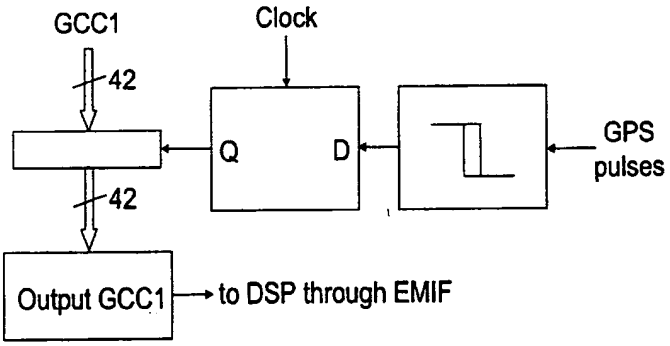


Figure 3-3: GPS pulses acquisition: hardware implementation

3.3 Correction application

The corrections are applied on the PN long code and short code generators on the transmit and receive sides of the BS through their code offset input. They are summarized on Figure 3-4.

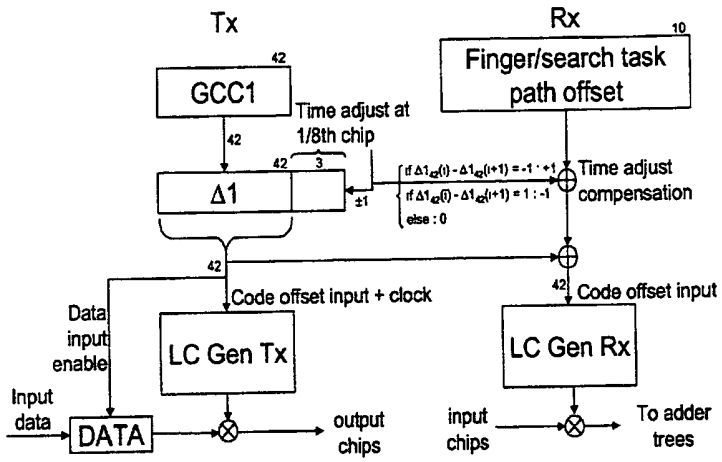


Figure 3-4: Time correction management on Tx and Rx sides

Δ_1 is a register that contains the chip offset value between GCC1 and the System Time Long Code. Thus, GCC1 is added to Δ_1 before controlling the code offset inputs of the Tx and Rx generators. Consequently Δ_1, as GCC1, is common to Tx and Rx sides. Though the range in chips of the offset to the Long Code fits into 42 bits, this register has 3 additional chips to provide an offset accuracy up to one 1/8 chip. This is due to the IS-95-A specifications that require a rate of change below 1/8 chip / 200ms on the Tx side [2],

[3]. The $1/8^{th}$ chip time adjust is provided by adding/subtracting $1/8^{th}$ chip corrections on the 3 LSB of Δ_1 while the Long Code generator is still controlled by the 42 MSB: Each increment of $(GCC1+\Delta1)_{42}$ drives a new data read and a PN generation.

On the Rx side, Δ_1 is only used to initialize finger and search tasks. While running, the finger tasks are synchronized by their own Delay Lock Loop functions, and the search tasks don't want to be interfered by any inner synchronization. Due to this, once these tasks are running, the time adjusts occurring on Δ_1 and leading to a roll over by 1 full chip are automatically compensated by incrementing/decrementing the path offset register of the finger/search task.

Two different kinds of corrections are applied:

- Power-up: Δ_1 offset register initialization.

- Tracking: Δ_1 offset register update.

3.3.1 Initialization

At power-up, the Δ_1 offset register is initialized with the value $\hat{\Delta}_1(i)$ given by the first received GPS pulse. As $\hat{\Delta}_1(i)$ is not an integer, Δ_1 is initialized to the nearest $1/8^{th}$ fraction of a chip using its full 45 bits range.
A dedicated task may not be required to initialize the registers Δ_1 and Δ_2 if the DSP can write directly into them.

3.3.2 Tracking algorithm

The tracking algorithm is elementary: in the steady state, the number of chips between two pulses k seconds apart is constant and equals kF_c. The DSP just needs to watch the variations of GCC1(i) – GCC1(i-k) compared to the constant value.
When [GCC1(i) – GCC1(i-k)] – kF_c > 1, the DSP will schedule eight decrements of the register Δ_1.
When [GCC1(i) – GCC1(i-k)] – kF_c < -1, the DSP will schedule eight increments of the register Δ_1.
All $1/8^{th}$-chip corrections (*increments or decrements*) shall be 200 ms apart as dictated by the standard.
The CCP provides Δ_1 and Δ_2 adjust capability by mean of an increment/decrement task associated to each register (see Section xxx).
How often should the tracking algorithm be run? With an internal clock providing sufficient stability in order to fulfill the IS-95 standard without use of external frequency reference: below 0.05 ppm over a long period (several days), the synchronization to GPS within ±3µs leads to a ±1 chip correction every minute (maximum rate). Scheduling the synchronization estimation tasks every 30 seconds will then be enough. The time between the measurement and the actual correction doesn't need to be short nor precisely defined: the short-term internal clock stability is such that the synchronization estimates remain available over several milliseconds. The (relaxed) synchronization between the estimation task and the correction task prevents generation of a (new) estimation before the previous correction (if needed) has been applied.

3.4 Absence of GPS (optional)

It might be required that the BS be operational when the GPS is not available at power up. In that case, the internal clock should provide a date/time clock powered by battery used to maintain the Local Time Base update when the BS is shut down. At power-up, GCC1 and GCC2 are always reset and Δ_1 and Δ_2 are initialized with values derived from the last available Δ_1 and Δ_2 offset values and the current date/time provided by the internal clock. Then, they are synchronized on the GPS if available as described above.

4. Operation Overview

This section describes CCP operations: the concept of cycles and their management, programming the CCP, management of tasks, output memory, and interrupts.

4.1 Tasks

The CCP executes host-specified tasks. In a CCP iteration, which lasts 16 chips, all running tasks process 16 chips of data. Then in the next CCP iteration, the same tasks continue the processing of the next 16 chips of data. Hence, a task typically runs for many CCP iterations. The CCP iteration boundaries occur when the Global Chip Count (GCC) is at a 16-chip boundary, that is, when GCC2 modulo 16 is 0.

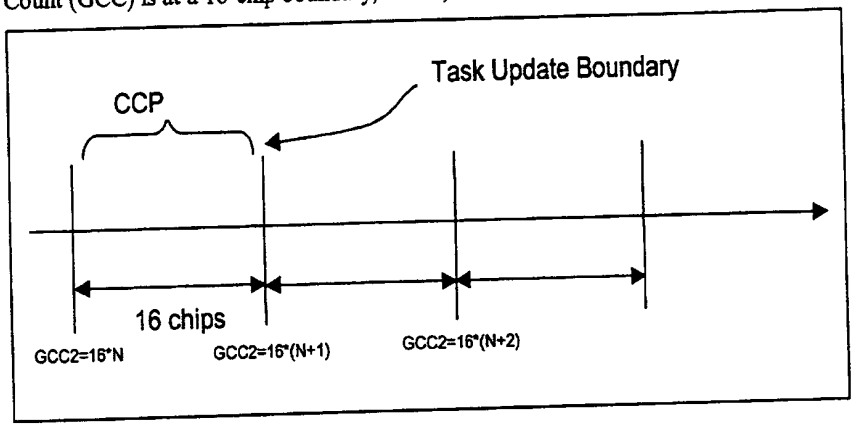


Figure 4-1: CCP Iteration

There are MAX_CYCLES CCP cycles available in each CCP iteration for the Rake Datapath to execute tasks PAR_SEARCH*MAX_CYCLES for the Search Datapath). The Rake Datapath cycles can be used for finger tasks (including DLL and specific pilot processing). The Search Datapath cycles can be used by search tasks such as DPE and R-ACH preamble detection. If there are extra cycles remaining, the CCP will automatically enter a power saving state. In each CCP iteration, most tasks consume more than one CCP cycle. The number of cycles required by each task is described along with each task in Section 6.3. New tasks are swapped-in and/or current tasks are modified at particular CCP iteration boundaries called Task-Update boundaries. Immediately following the Task-Update boundary, the CCP swaps the tasks, as requested, into the Task Buffer, then starts executing tasks from the first address of the Task Buffer. Each Task-Update boundary is associated with the Task-Update Interrupt event. Details are presented in Section 6.1.

DPE/Search tasks require considerably more than one iteration pass to complete, and stop when they are completed. Finger tasks run continuously until they are disabled by the host processor.

4.1.1 Cycle Management

It is the responsibility of the software running on the host processor to manage the available MAX_CYCLES CCP cycles. If the MAX_CYCLES cycles are expended and the CCP has not executed all enabled tasks, an Error interrupt event is issued. In order to assist in cycle management, and for debugging purposes, the Cycle_Count register is updated every CCP iteration by the CCP to indicate how many cycles were expended in the last CCP iteration. This register will change infrequently as tasks are added or completed. To ensure that an accurate value is read from this register, it should be read directly following the Task_Update boundary.

4.1.2 Task Management

Up to MAX_FINGERS/MAX_DPE tasks can be stored in the Task Buffer. They are acted on in numerical order by address. The order of placement of tasks in the Task Buffer does not affect the results of a task in any way. The CCP processes only the tasks that are "enabled," as discussed in Section xxx. After the CCP finishes executing all of the enabled tasks, its data path will be shut down for the remaining cycles in the iteration, and reads and writes to data path memory will cease. Some of the control logic will remain operational. This will result in significant power savings within the CCP.

4.1.3 Adding New Tasks

New tasks may be added at any time by writing them to the Task Buffer and then requesting an immediate load for that task. The configuration memories and registers must be initialized before a task, which references them, is programmed. A task may actively run in the first CCP iteration that follows the load of a task into the hardware side of the Task Buffer. Therefore, configuration parameters for a task must be programmed before, or *on* the Task_Update boundary that loads the task.

4.1.4 Starting and Stopping Tasks

Having the task specification in the Task Buffer is not enough for the task to execute, the task must also be enabled by requesting either a synchronous start or an immediate start. An immediate start takes effect at the Task-Update boundary on which the request is received, and a synchronous start takes effect at the slot number specified for that task (which is with reference to the task's own slot timing). Each task can be enabled at any Task-Update boundary. There is no limit on the number of tasks in the Task Buffer that may be enabled, except in the total cycles consumed by the ones enabled.

The same concept applies to stopping tasks. All tasks can also be requested to stop immediately or synchronously at any Task-Update boundary.

4.1.5 Modifying Running Tasks

The mechanism to modify tasks is the same as loading tasks. To change a task's parameters, the whole task specification must be written into the Task Buffer and then an immediate load requested; no change to the enable status is made. In addition, a task may be synchronously (to its own slot timing) be reloaded.

Changing task parameters must be done with great care because the new parameters may conflict with the previous ones, thus producing erroneous results.

4.1.6 Task Run/Stop Status

The CCP keeps a status register indicating the run/stop state of each task. A task is in one of four states: stopped, waiting to run, running, or waiting to stop. An "enabled" task is one that is running, waiting to run, or waiting to stop.

Once the task begins executing, it will execute forever (until stopped by SW) or until a well-defined end time. The stopping of a task can be done by software manually, or will be done by the CCP automatically in the case of a task with a well-defined end-time. Automatic de-activation (stopping) times are described along with each task's description in Section 6.3. A "disabled" task is one that is in the stopped state.

4.2 Configuration Parameters

Configuration parameters are information used by running tasks. Some parameters are expected to be changed while a task is running, but most are not. There are provisions to allow changes to those parameters, which may need to change while the task is running so that the changes have predictable results. The parameters, which have such provisions, are the Finger Interrupt Table, the Finger Symbol Buffer Configuration Table, and the Walsh Table.

Entries in the Finger Interrupt Table are transferred from the software side of the double buffer, to the hardware shadow of the double buffer at each Task_Update boundary.

Entries in the Walsh Table and Finger Symbol Buffer (FSB) Configuration Table, may be written to at any time. Pointers to the tables are used by the tasks, and it is these pointers which can be changed while the task runs with predictable behavior. It is the responsibility of the software to avoid writing to any entry, which is currently in use by the CCP.

4.3 Initializing the CCP

After a power-on reset, all the CCP registers are in a known state, the CCP memories have contents which are undefined, and the CCP itself is not running any tasks. In order to configure the CCP to run its first tasks, the following sequence of steps is recommended, once the clock of the CCP has been started:

1. System Time acquisition (see Section 3).

2. Configuration memories and registers should be programmed (e.g. Walsh Table, Finger Symbol Buffer Configuration Table, Finger Interrupt Table, Task_Update_Cycle register, Pilot Bits Table, Interrupt etc.)

3. Tasks should be written to the Rake Task Buffer and Search Task Buffer.

4. Task Requests (Start, Load) should be written to their respective registers.

5. Then the CCP should be started with the Start/Continue command.

At the first Task_Update boundary, all specified Task Requests (Start, Load) will occur and the tasks will begin running.

The content of all memories is undefined and must be initialized before being referenced. The entire Task Buffer needs not be initialized before use, just the locations, which are to be used. All tasks are disabled at reset.
Task Update interrupts begin when the CCP is enabled to run via a start command or by a sleep timer.

4.4 Finger Symbol Buffer Management

The Finger Symbol Buffer is set up as multi-slot circular buffer areas for each despreader (sub-task) which uses them, as specified in the FSB Configuration Table. It is the responsibility of the software running on the host processor to manage these areas, and in keeping the areas from overlapping.

When the CCP is writing to a particular address in this memory, the reading of that address would result in an undefined data word being read. For this reason, the SW must react to interrupts in a timely way so as to always read in "safe areas". Further details are specified in Section 6.1.6.1.

4.5 Interrupts

Many events in the CCP may cause interrupts on one of the external interrupt lines. There are system interrupt events, error interrupt events, and task-based interrupt events. Each interrupt event can be independently. A FIFO queuing system is implemented for the task-based interrupts, so that several pending interrupts can be held until the processor services them. This is described further in Section xxx.

5. CCP Global Control

5.1 Task Management

All tasks have an ID associated with them, which is used by the CCP to identify which configuration parameters are associated with them and where the results of the tasks are to be placed. The same ID should not be duplicated on two tasks of the same type, which run simultaneously. The results of such a situation are not defined.

There are two types of task-level ID's:

- Finger ID
- DPE/Search ID

The Finger ID/UserID combination identifies a Finger Task among others Finger Tasks. It is used by the DSP to determine the Finger Task origin of a slot interrupt. (See section xxx).

The FingerID/WalshID combination is used to index FSB Configuration Table, FSB and EOL Buffer.

Any running DPE/Search tasks must have a unique DPE/Search ID's. The DPE/Search ID of a task is used to index internal scratch memories, and DPE Buffer. It is also used to distinguish different DPE task events at the interrupt FIFO's.

In each Walsh block, each Walsh sub-task has a unique Walsh ID to distinguish itself from other Walsh sub-tasks.

5.1.1 Task Buffer and Associated Configuration Memories

Each Data Path (Rake and Search) provides a Task Buffer in which the DSP writes tasks instructions.
Tasks are written to the Task Buffer by software, and after they are loaded into the hardware side, they can be executed by the CCP. The loading is requested by software using the Task Request bits along with a Task Request ID, and occurs at the next Task_Update boundary following the write of the request. This is the basic mechanism for loading tasks into the CCP for execution.

Tasks Update, which are common for all tasks, are defined in the CCP Internal Time Base, provided by GCC2.
When the Task_Update transfer takes place, the Task_Update_Timestamp register is loaded with the current value of the global chip counter (GCC2). The time interval for the Task_Update boundary is programmable. It must be set at a multiple of the CCP iteration (16 chip periods). This multiple of 16 chips is stored in the double-buffered shadow register Task_Update_Cycle, which is updated by copying the software side of

the register to the hardware side on every Task_Update boundary. At reset, the value of Task_Update_Cycle is 4.

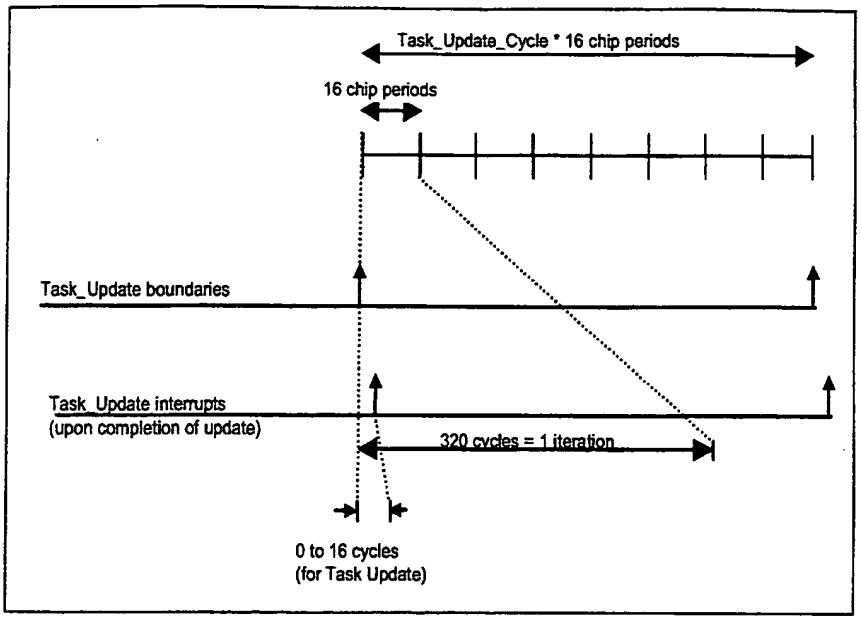


Figure 5-1: Task Update Timing

The DSP may read the contents of the Task Buffer. After the Task_Update interrupt occurs, which signals the completion of the Task_Update actions, reading the Task Buffer will not result in a synchronization problem. See Figure 5-1.
After power-on reset, the contents of the Task Buffer are undefined, the enable status bits of all tasks in the execution buffer are disabled, and all Task Request bits are cleared. Tasks will begin being executed once the tasks are written to the Task Buffer, Task Requests to load and start are made, and the CCP is started.

5.1.2 Starting and Stopping Tasks

The CCP maintains a Task Start/Stop Status register, which reflects which tasks are "enabled." Enabled tasks are those that are "Running", "Waiting to Run", or "Waiting to Stop". The CCP only fetches a task if it is enabled. Hence, disabled tasks are skipped over and do not expend CCP cycles.
DPE tasks are autonomously stopped by the CCP when completed. The DSP must stop finger task. The CCP cannot autonomously start any tasks.

The DSP requests the starting and stopping of CCP tasks at the Task_Update boundary. All tasks may be started or stopped on an individual basis. The DSP sets the appropriate Task Request Bit and writes the type of request to the Task Request ID register for the task.

A task that is enabled may or may not be actively running; it depends on whether its start time has been reached. If the start time has not been reached, then the enabled task is "Waiting to Run", and one CCP cycle is expended per task. A task which is "Running" or "Waiting to Stop" is actively running and may consume several CCP cycles.

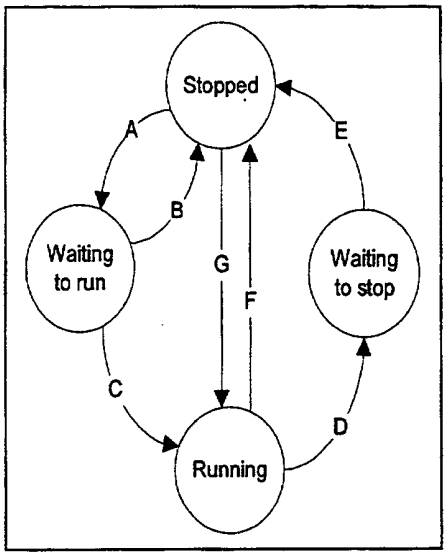


Figure 5-2: Task Start/Stop Transition Diagram

Description of task state transitions:

A- From Stopped to Waiting-to-Run: When a synchronous start is requested.
B- From Waiting-to-Run to Stopped: When immediate stop is requested.
C- From Waiting-to-Run to Running: When slot activation time arrives.
D- From Running to Waiting-to-Stop: When synchronous stop is requested.
E- From Waiting-to-Stop to Stopped: When slot activation time arrives or immediate stop is requested.
F- From Running to Stopped: When immediate stop is requested or task completes.
G- From Stopped to Running: When immediate start is requested

5.1.3 Synchronous Task Reloading

A Task may be synchronously reconfigured with reference to its own slot/frame timing. A Task can start, stop, or reload configuration on slot boundaries. This is

accomplished by requesting the appropriate action for that task and identifying the slot boundary for that action to occur.

5.1.4 Task Start Time

All CCP tasks may be started immediately, on the very next slot boundary at which the request is made, or synchronously to its own slot timing. The starting slot may be radio slot 0 through slot 15. The starting slot is defined in the finger frame reference (see Figure 6-22).
A task which is "Waiting to Run" expends one CCP cycle until it enters the "Running" state, at which time it uses the number of CCP cycles determined by its specifications. In some situations, an immediate start may be desirable – the Finger task for example. For other tasks, however, an immediate start will result in erroneous results.

5.1.5 Task End Time

Some tasks execute without end, others have well defined end times.

- Finger: continuous
- DPE/Search: one-shot

5.2 Command Set

All commands described in this section are common to the Rake and Search Data Path. The following are memory-mapped commands: when SW writes to the address, the command is effective immediately, reading the address has undefined results.

5.2.1 Start / Continue Command

The Start/Continue command is used to start the CCP after power-on (see Section 4.3), and to continue the execution of the CCP after a Halt command. The CCP will always start on a 16-chip boundary of the Global Chip Counter (GCC). The step value defines the number of cycles to run before stopping, a value of "0" indicates to run continuously.

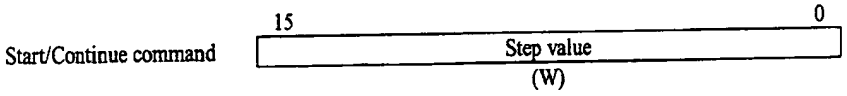


Figure 5-3: Start/Continue Command

5.2.2 Halt Command

The Halt command forces the CCP to halt on the next 16 chip boundary. Processing can be resumed with the Start/Continue command but there will be a break in the input stream which must be accounted for by SW. For proper operation, the procedure for programming the CCP after power-on should be followed (see Section 4.3).

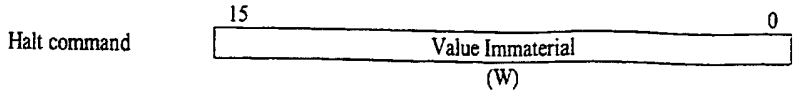


Figure 5-4: Halt Command

5.2.3 Software Reset Command

When the CCP is stopped or halted, this command will reset all internal registers and states to the power-on reset configuration. This command is not intended to be used while the CCP is executing.

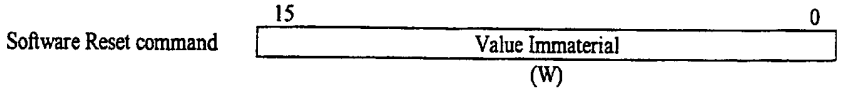


Figure 5-5: Software Reset Command

5.2.4 (Δ1 & Δ2) Increment/Decrement Command

Δ1 & Δ2 Registers respectively contain the chip offset value between GCC1 and between the System Time Long Code and between GCC2 and the System Time Short Code.

This command is used to increment by 1/8 of chip (or decrement by 1/8 of chip) Δ1 & Δ2 register. When Inc/dec_ bit is set to "1", Δ1 & Δ2 are incremented by one chip/8, when Inc/dec_ is set to "0", Δ1 & Δ2 are decremented by one chip/8.

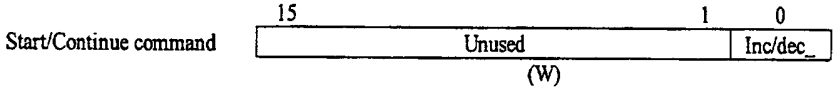


Figure 5-6: (Δ1 & Δ2) Increment/Decrement Command

5.3 Global Status

All registers described in this section are common to the Rake and Search Data Path. A synchronizer circuit allows the processor to read accurate values from these registers.

5.3.1 CCP_Status Register

The CCP_Status Register contains 1 bit. "1" if the CCP is running, "0" if stopped.

Figure 5-7: CCP_Status Register

5.3.2 Task_Update_Cycle Register

The Update_Cycle register identifies the how often there is a Task_Update event. It is implemented with a shadow register that is updated on the Task_Update boundary. At reset, the value of Update_Cycle is 4 decimal.

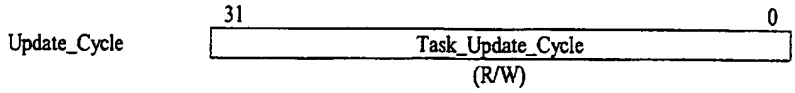


Figure 5-8: Task_Update_Cycle Register

5.3.3 Task_Update_Timestamp Register

The Task_Update_Timestamp Register captures the GCC2 value of the most recent Task_Update boundary.

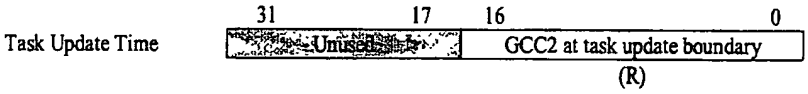


Figure 5-9: Task Update Time register

5.3.4 Int_System_Event_Status Register

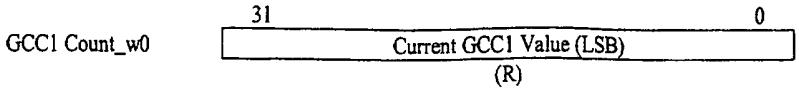
Reading the Int_System_Event_Type clears the System interrupt event.

- Task_Update Event. 1 bit. "1" if the Task_Update event occurred, "0" otherwise.

Figure 5-10: Int_System_Event_Status Register

5.3.5 GCC1 & GCC2 Count Register

The GCC1 & GCC2 Count Registers capture the current value of GCC1 & GCC2 value, updated once per chip. GCC1 & GCC2 Count register are read only.



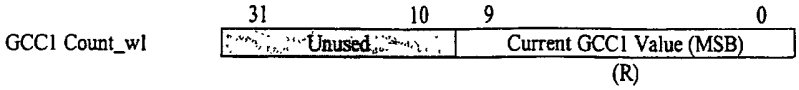


Figure 5-11: GCC1 Cycle count register

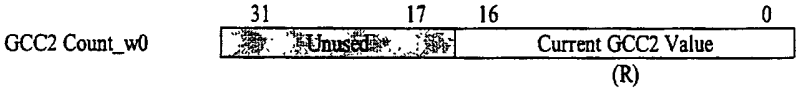
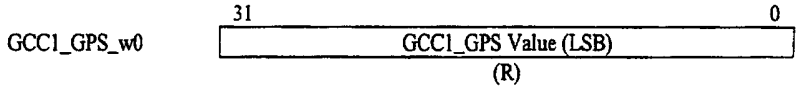


Figure 5-12: GCC2 Cycle count register

5.3.6 GCC1_GPS & GCC2_GPS Register

GPS pulses are used to push the current values of GCC1 and GCC2 into GCC1_GPS and GCC2_GPS registers respectively.



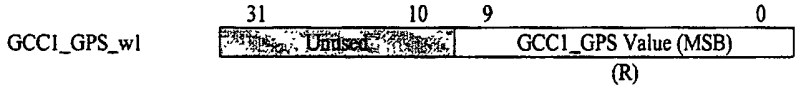


Figure 5-13: GCC1_GPS register

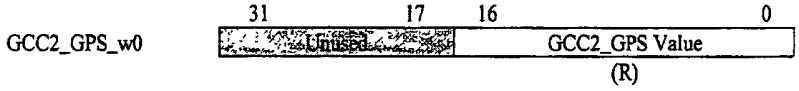
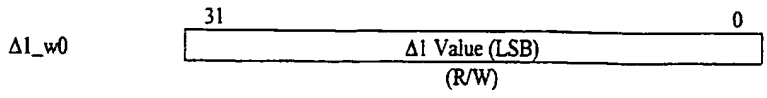


Figure 5-14: GCC2_GPS register

5.3.7 Δ1 & Δ2 Registers

Δ1 & Δ2 Registers respectively contain the chip offset value between GCC1 and between the System Time Long Code and between GCC2 and the System Time Short Code.

These offset values are computed by the DSP (see section 3) using the GCCx_GPS registers value and the associated time information from the GPS receiver and should be initialized by the DSP at power-up. Then, the tracking (Δ1 & Δ2 increment/decrement) is performed using Δ1 & Δ2 increment/decrement command. (see section 5.2.4).



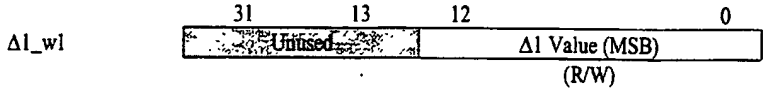


Figure 5-15: Δ1 register

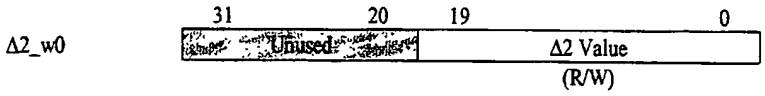


Figure 5-16: Δ2 register

6. Rake Data Path Control

6.1 Software Interface

This section describes all registers and all configuration tables that the host processor has to access in order to either configure the Rake Data Path or get the Rake Data Path status information.

6.1.1 Rake Task Buffer

The Rake Task Buffer is a collection of MAX_FINGERS ping/pong buffers each of which may contain a CCP task of 64 bits. The status of each of MAX_FINGERS tasks as to whether the ping or pong is available to the CCP for execution, and the other available for SW loading/modification, can be controlled on an individual task basis by SW. Task formats are found in Section 6.2. Host processor may read entries modifiable by the SW.

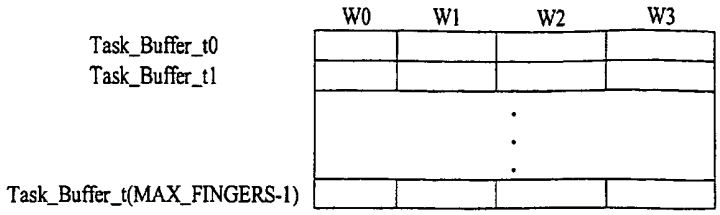


Figure 6-1: Rake Task Buffer

	15 (R/W) 0
Task_Buffer_t0_w0	First Word of Task Buffer Entry (MSW)
Task_Buffer_t0_w1	Second Word
Task_Buffer_t0_w2	Third Word
Task_Buffer_t0_w3	Last Word of Task Buffer Entry (LSW)

Figure 6-2: Rake Task Buffer Entry Description

6.1.2 Synchronously Updated Configuration Parameters

These configuration parameters are double buffered and are updated at the Task-Update boundary: the software-modified register is copied to the hardware-readable shadow register at this time.

6.1.2.1 Rake Task Request Bits

Task Requests to Start/Stop, Load/Reload, or Adjust Timing may be made at the next Task-Update boundary (following a write access) by setting the appropriate Task Request Bits. A "1" indicates a new request, '0' indicates no new request (requests may still be pending). The requesting will continue at succeeding Task-Update boundaries until the request is cleared by SW, via a "0", therefore, for proper operation, a "0" should be written at the Task-Update boundary immediately following a "1" being written. Refer to Figure 6-1. An operational description of Task Request is found in Section 5.1.2.

	15 (R/W) 0
Task_Req_w0	Request for tasks 15-0
Task_Req_w1	Request for tasks 31-16
Task_Req_w7	Request for tasks (MAX_FINGERS-1) – (MAX_FINGERS-16)

Figure 6-3: Rake Task Buffer Entry Description

Writing the appropriate information to the task's Task Request ID field indicates the type of request.

6.1.2.2 Rake Task RequestIDs

Each task has a Task Request ID register which identifies the type of Request being made when a "1" is written to the corresponding Task Request Bit. The format of the Task Request ID is:

	9	8 7	6	5	4	3 0
Task Request ID	Start/Stop Enable	Start/Stop ID	Load/Reload Enable	Load/ Reload ID	Timing Adjust Enable	Slot#
			R/W			

Figure 6-4: Task Request ID Register Format

- **Start/Stop Enable**: a "1" indicates that the request includes the start/stop action identified in the Start/Stop ID bits, a "0" indicates no start/stop action is requested.

- **Start/Stop ID**: These bits are referenced by the hardware only if the Start/Stop Enable bit is set ("1"). They have the following meaning:
  - "00" indicates Immediate Start
  - "01" indicates Synchronous Start
  - "10" indicates Immediate Stop
  - "11" indicates Synchronous Stop

- **Load/Reload Enable**: a "1" indicates that the request includes a load/reload action as identified in the Load/Reload ID bit, a "0" indicates no load/reload action is requested.

- **Load/Reload ID**: a "0" indicates an Immediate Load is requested, a "1" indicates a Synchronous Reload is requested. This bit is referenced by the hardware only if the Load/Reload Enable bit is set ("1"). A synchronous Reload implies the instruction is "enabled", if it is not, the Reload will remain pending until an Immediate Load is requested.

- **Timing Ajust Enable** (for Finger Tasks only): a "1" is a request to adjust timing, and "0" is not a request to adjust timing. The timing change information is contained within the Finger Task description.

- **Slot #**: This field identifies the slot number for a synchronous action (synchronous start, synchronous stop, or synchronous reload). When a synchronous start is requested, this field defines the first slot to begin processing. When a synchronous

stop is requested, this field defines the first slot which is not processed, in other words, the last slot number processed is the one, which precedes this slot number (modulo 16). When a synchronous reload is requested, this field defines the first slot after the reload, in other words, the last slot number processed before the reload is the one which precedes this slot number (modulo 16). The sixteen slots per frame are numbered 0 through 15.

6.1.2.3 Rake Walsh Table

The Rake Walsh Table holds Walsh sub-tasks. A set of Walsh sub-tasks is used by a Finger task to specify how Walsh de-spreading should take place, on one or more (in case of Finger) Walsh channels. Each Finger task has a Walsh Pointer field to specify the desired set of Walsh sub-tasks. In addition, Finger tasks may specify the processing of an EOL sub-task, which will support the DLL function by computing on-time, early and late energies on a particular Walsh channel.

The Rake Walsh Table is made up of (MAX_USERS + 16)[7] different sections or sets, called Walsh blocks, each configured with 8 Walsh sub-tasks. A unique UserID points each Walsh block. Up to MAX_FINGERS_PER_USERID Finger Tasks can point to the same Walsh block.

The Figure 6-5 describes the Rake Walsh Table structure.

[7] If the information contained in the a Walsh blocks needs to be changed (e.g. Adding a new Walsh code) while the finger task is running, we just set up an unused Walsh Block and then change the Walsh pointer in the Finger task instruction.

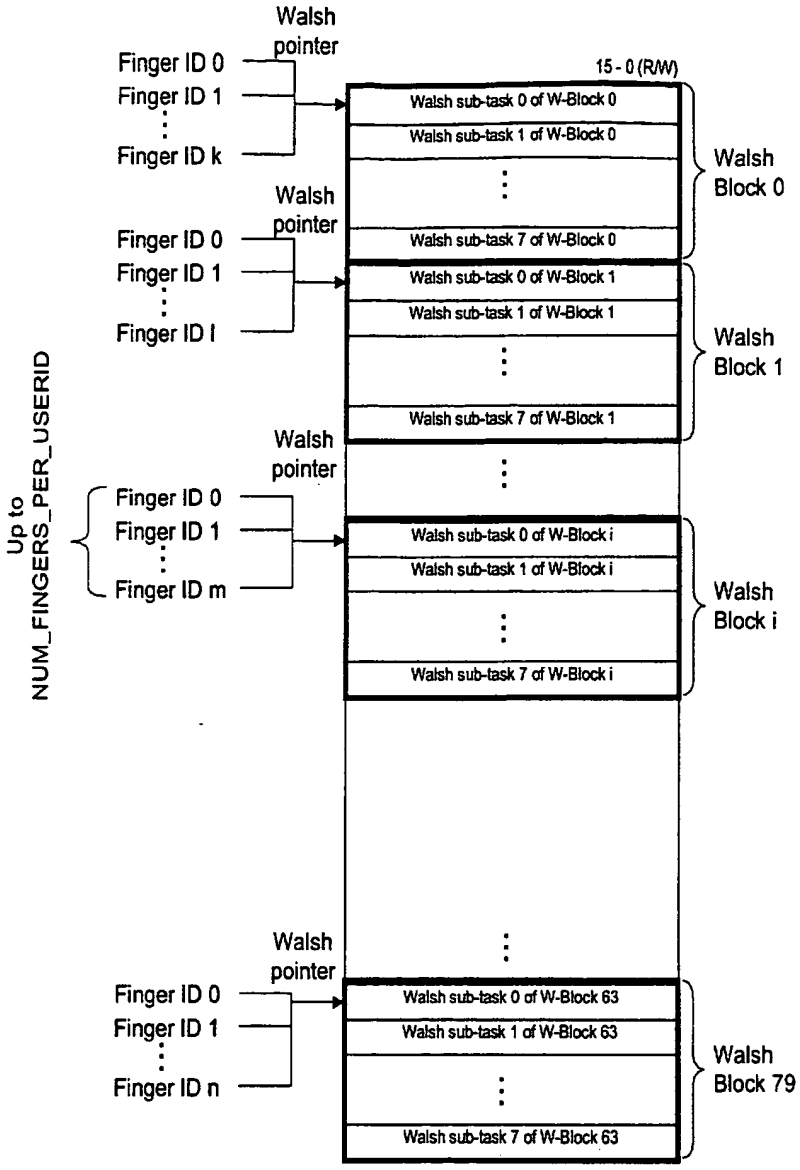


Figure 6-5: Rake Walsh Table structure

If less than the maximum number of entries (8) is needed, the sub-tasks must appear at the beginning of the area and be consecutive. If there is an EOL Walsh sub-task, it must be at the first location.

Each sub-task has a Walsh ID, which must be unique in the set. The Walsh ID is chosen by the host processor and allows for flexible reconfiguration of Walsh sub-tasks while a Finger task is running.

The Walsh Table is directly written in by the DSP. A sample Walsh sub-task is shown in Figure 6-6 that belongs to the fifth Walsh block (S5) and is the third Walsh sub-task (ST3).

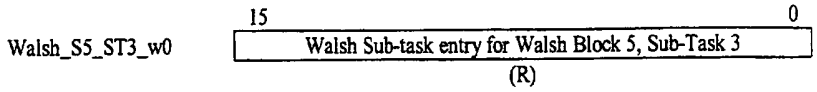


Figure 6-6: Walsh Table Entry

The Figure 6-7 summarizes the registers and tables the DSP must set to specify a task.

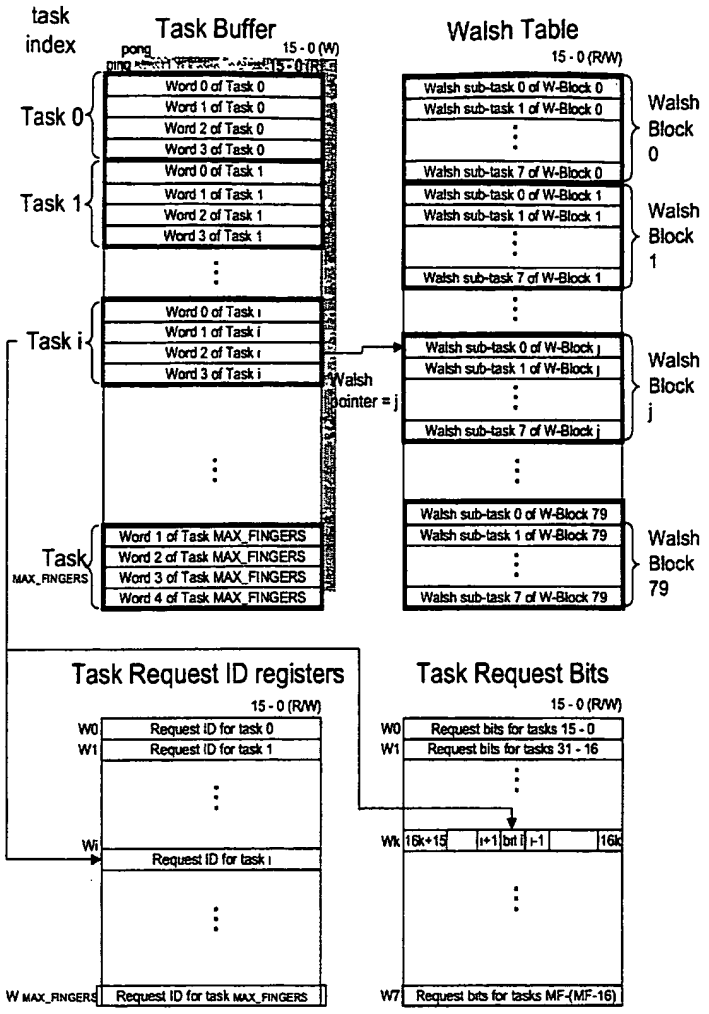


Figure 6-7: Registers and tables used to specify a task

6.1.2.4 Finger Symbol Buffer Configuration Table

The FSB Configuration Table is a ping/pong buffer with two entries for each Finger ID/ Walsh ID (despreader) combination, to specify where in the Finger Symbol Buffer a finger and each of its Walsh channels (as specified by the Walsh ID) will output its de-spread symbols. The Finger Symbol Buffer Configuration Table contains 8*MAX_FINGERS entries. Each entry consists of the start address for the first slot of data in the four-slot circular buffer and the address offset from one slot to the next slot.

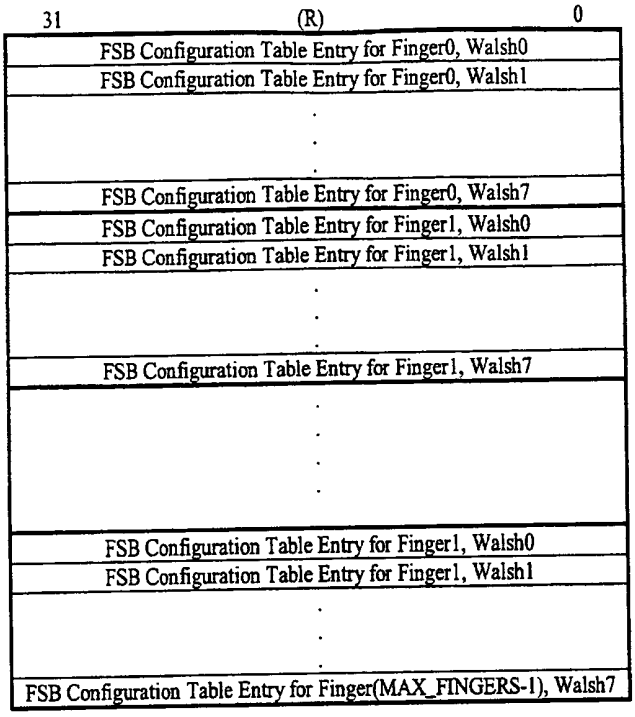


Figure 6-8: Finger Symbol Buffer configuration Table

The FSB Configuration Table is written to directly, with the ping and pong sides written independently. Each entry is 32 bits wide. The entries are readable as shown in Figure 6-9.

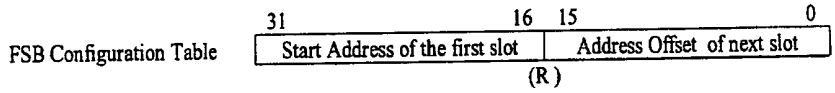


Figure 6-9: FSB Configuration Table Entry

6.1.3 Asynchronously Updated Configuration Parameters

Asynchronously updated configuration parameters are typically ones that are not changed often, if at all, once the CCP begins its operations. Changing these parameters when not in use is not a problem.

6.1.3.1 Rake External Interrupt Enable Register

The Rake External Interrupt Control register controls the enabling of FIFO, error and system interrupts to the external interrupt lines.

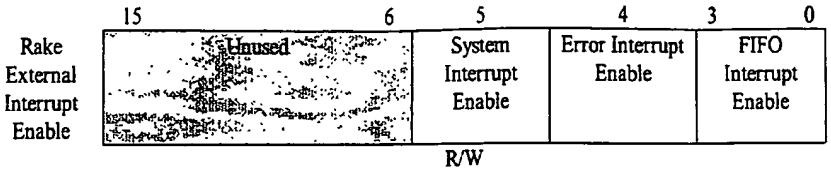


Figure 6-10: Rake External Interrupt Enable Register Format

6.1.4 Rake Data Path Status

6.1.4.1 Task Ping/Pong Status Bits

The host processor can read the current ping/pong status for all tasks in Task Buffer. "0" indicates that HW is reading ping and SW reading/modifying pong, "1" indicates the reverse.

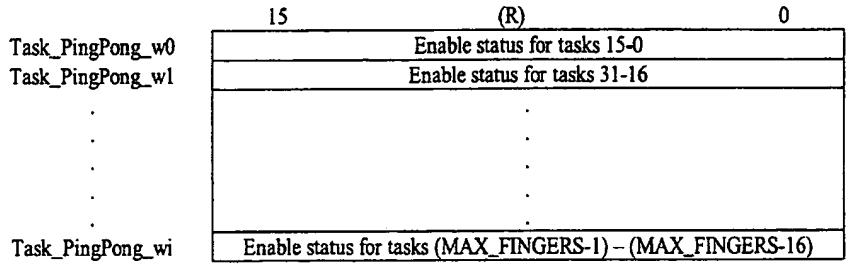


Figure 6-11: Task Ping/Pong status Bits

6.1.4.2 Task Run/Stop Status Bits

The host processor can read the current Run/Stop status for all tasks in Task Buffer.

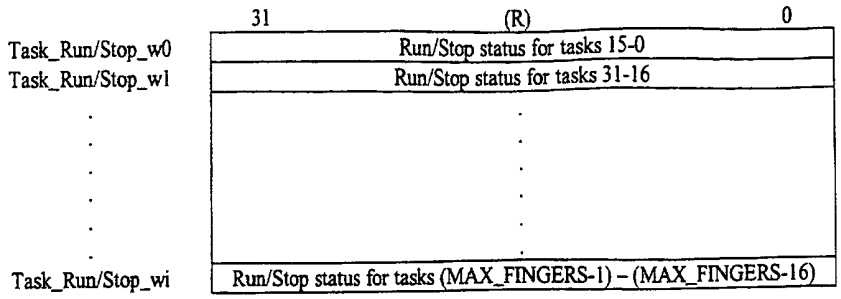


Figure 6-12: Task Run/Stop Status Bits

Each task has a two bit field which encodes the Run/Stop status in the following manner:

- "00" indicates "Stopped"
  - "01" indicates "Waiting to Run"
  - "10" indicates "Running"
  - "11" indicates "Waiting to Stop"

6.1.4.3 Cycle_count Register

The Cycle_Count register captures the number of cycles expended in the most recent CCP iteration of the Rake Data Path; it is updated every CCP iteration.

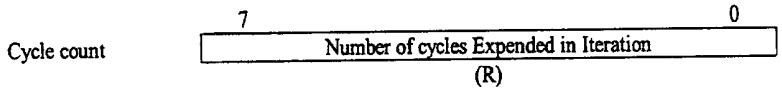


Figure 6-13: Cycle count register

6.1.5 Interrupt Status

A synchronizer circuit allows the processor to read accurate values from these registers.

6.1.5.1 Rake_Int_Error_Event_Status Register

Reading the Rake_Int_Error_Event_Status clears the Error interrupt event.

- Cycles Exceeded Error. 1 bit. "1" if the number of cycles attempted in an iteration was greater than the maximum (MAX_CYCLES), "0" otherwise.
- Interrupt FIFO Overflow Error. 4bits. One bit for each of the four interrupt FIFO's, "1" if the corresponding FIFO has overflowed, "0" otherwise.

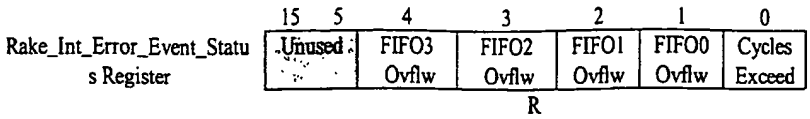


Figure 6-14: Rake_Int_Error_Event_Status Register

6.1.5.2 Rake FIFO Status & Content

The RAKE FIFO Status register shows each interrupt FIFO's empty/non-empty status. When a particular FIFO is not empty, its FIFO Empty Status is "1". The status bit is cleared when all FIFO contents have been read out.

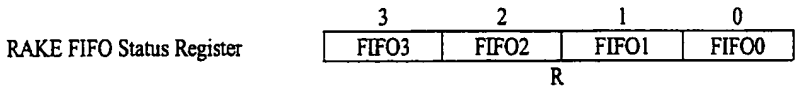


Figure 6-15: RAKE FIFO Status Register

RAKE FIFO contents are 2 words each, and are read out at one memory location, one after the other. When the second word is read out, the FIFO hardware increments its internal read pointer.

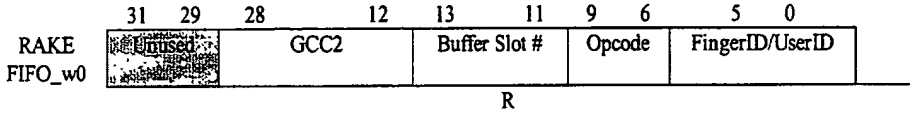


Figure 6-16: RAKE FIFO Content Format

The Finger task may issue multiple and simultaneous interrupt events for the slot, TPC, and Pilot, which are indicated by individual status bits in the first word. The Buffer slot number field indicates which slot in the multi-slot circular buffer that a Finger task or Search task has placed its data; field is valid on when the Slot Interrupt bit field is set for Finger task, and on all Search task interrupts.

The value of GCC2 is recorded in GCC2 field when the event occurred.

6.1.6 Output Data

6.1.6.1 Finger Symbol Buffer (FSB)

The Finger Symbol Buffer holds <*TBC*> 32 bit words and is divided into multi-slot circular buffers for each Finger ID/ Walsh ID combination. The location of each circular buffer is defined in the FSB Configuration Table. Refer to Section xx. When a Finger Task has a spreading factor between 8 and 512 the following is the format of the data:

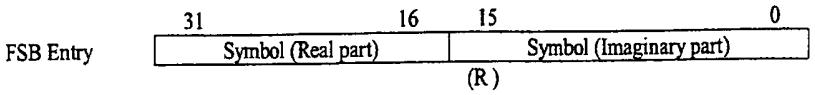


Figure 6-17: FSB Entry Format

When a Finger Task has a spreading factor of 4, the following is the format of the data:

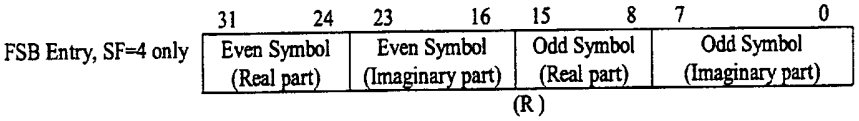


Figure 6-18: FSB Entry Format, SF=4 only

6.1.6.2 Finger Max Buffer

The Finger Max Buffer stores the largest energy value within a slot for a particular Finger ID/ Walsh ID (despreader) combination in four slot circular buffer. Each energy value is 16 bits. There are 8*MAX_FINGERS four-slot circular buffers. They are arranged in the same order as the FSB Configuration buffer, and the four slot energies are grouped together.

The buffer number (0-3) used for a particular slot is the same as the buffer number used by the FSB to store symbols, and the Finger Task interrupt information provides this buffer number in the FIFO entry.

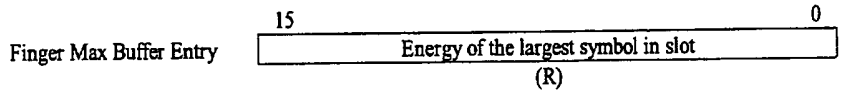


Figure 6-19: Finger Max Buffer Format

6.1.6.3 EOL Buffer

The EOL Buffer stores early, on time, late energy measurements of Finger tasks. Results are dumped into the EOL Buffer once per frame -- at the end of each frame -- and must be retrieved by the host processor before the next frame boundary. When new results are ready, they may be read by the DSP.

The Finger ID/UserID combination indexes the outputs, as shown below.

31 24	23 (R) 0
Unused	Early Energy for Finger0
Unused	On-time Energy for Finger0
Unused	Late Energy for Finger0
	. . . .
Unused	Early Energy for Finger(MAX_FINGERS-1)
Unused	On-time Energy for Finger(MAX_FINGERS-1)
Unused	Late Energy for Finger(MAX_FINGERS-1)

Figure 6-20: EOL Buffer

This section describes how tasks are transferred into the Rake Data Path, when they begin executing, and when they complete. It also describes the task set available.

The Rake Data Path supports Finger tasks, including support for EOL (early/on-time/late) measurement. This task is a continuous task.

6.1.6.4 Raw Pilot Buffer

The Raw Pilot Buffer stores the raw pilot symbols, or sub-symbols, despread in the Reverse Control Channel. (See section 6.2.1)

The Finger Symbol Buffer holds <*TBC*> 32 bit words and is divided into multi-slot circular buffers for each Finger ID/ UserID combination.

The buffer number (0-3) used for a particular slot is the same as the buffer number used by the FSB to store symbols, and the Finger Task interrupt information provides this buffer number in the FIFO entry.

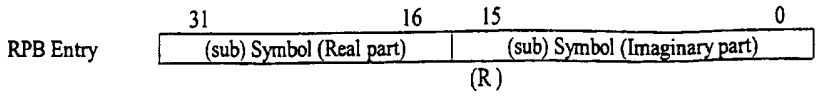


Figure 6-21: **RPB Entry Format**

6.2 Finger Task Description

6.2.1 1xRTT Finger task

The CCP supports up to MAX_FINGERS Finger tasks.

The finger task is used for despreading finger symbols and EOL measurement (DLL support) together with specific pilot channel despreading. Despread finger symbols are stored in the Finger Symbol Buffer. EOL energy measurements are output to the EOL Buffer. Despread pilot channel symbols (or sub-symbols) are output to the Raw Pilot Buffer.

Depending on which standard is used (IS95 or 1xRTT), different finger tasks are available.

The Finger tasks time parameters definition is illustrated in **Figure 3-2**.

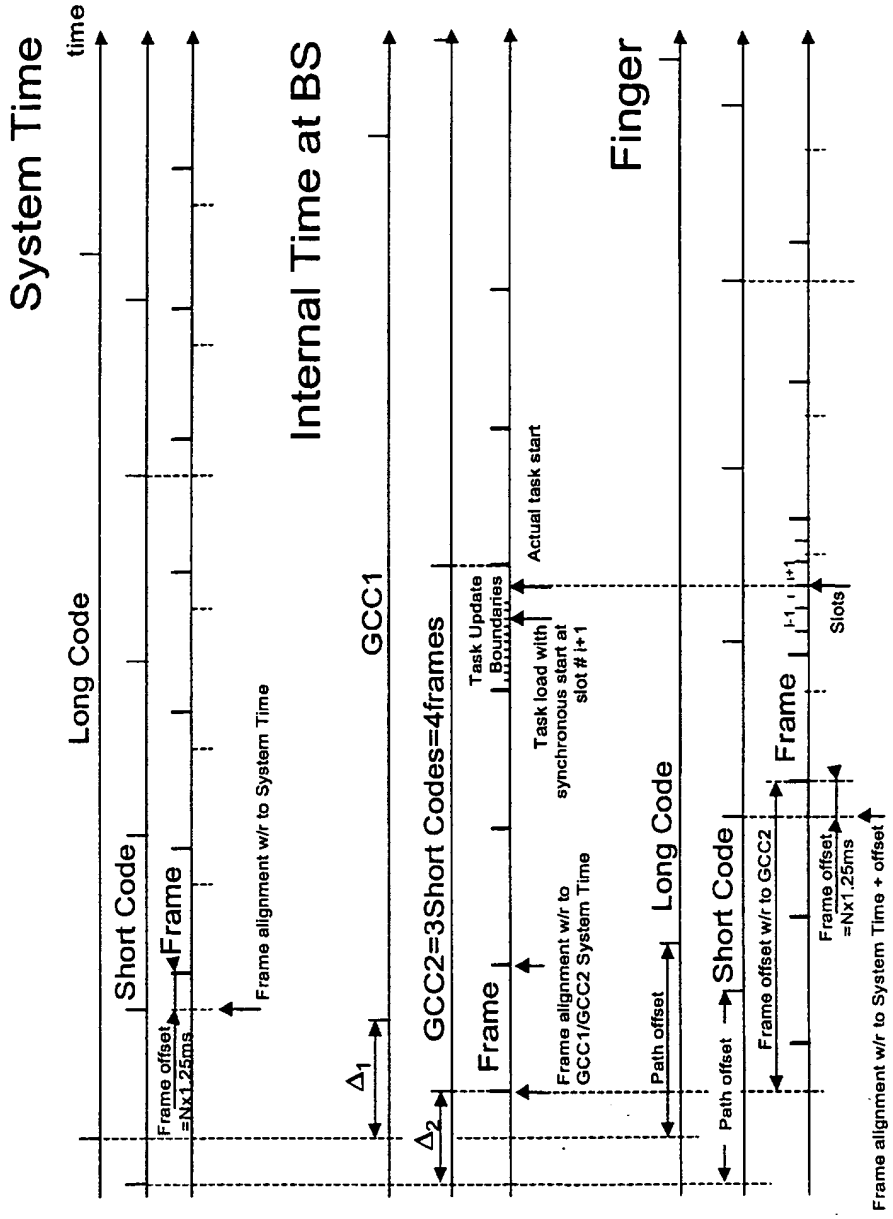


Figure 6-22: Finger task time parameters

The 1xRTT finger task is used for despreading 1xRTT fingers, EOL measurement (DLL support) and for providing raw pilot symbols to the DSP for channel estimation.

The finger task can generate one interruption every SLOT_SIZE*PS_SIZE chips.

EOL measurements may be done either on the Reverse Data Channel or on the Reverse Control Channel (Pilot/Data mode). This mode is selected in the associated EOL Walsh sub-task (see Section 6.2.2.3).

When operating on the data channel, EOL measurements are performed by integrating coherently chips over one symbol duration, then dumping continuously symbol energies. Symbol duration is defined by the Walsh sub-task SF parameter (Section 6.2.2).

When operating on the pilot channel, chips are integrated over coherent packets ((PS_SIZE/N_{CA}) chips if Mode0, or (N_{CA}*SLOT_SIZE) chips if Mode1)[8] then, the packets energies are dumped continuously. N_{CA} is a parameter of the Walsh sub-task (Section 6.2.2).

The non-coherent accumulation is reset on every frame and energy results are output once per frame in the EOL Buffer, updating the previous values. Consequently, in Mode 1, N_{CA} cannot be larger than the number of slots per frame, 16.
Also, when operating in Mode 1, if N_{CA} does not divide into the number of slots per frame (16), the non-coherent accumulation is processed on round-down (16/N_{CA}) energy accumulations.

When EOL_en bit is set and EOL measurements are processed on the pilot, the finger task performs a coherent accumulation on the pilot channel and raw pilot channel symbols are output every PS_SIZE /N_{CE} chips in the Raw Pilot Buffer (RPB). N_{CE} and N_{CA} are independent values.

When coherent Mode 1 is used, Skip_PC mode should be disabled in order to allow to perform the coherent accumulation of an EOL task on the full range of N_{CA}*SLOT_SIZE chips, when, for example, a finger is running on the R-ACH with EOL specified on the Reverse Pilot Channel[9].

[8] Coherent integration modes, Mode1 and Mode2, are defined in the associated Finger Walsh sub-task (see Section 6.2.2.2).
[9] In 1xRTT, the Reverse Pilot Channel transmitted along with the Reverse Access Channel doesn't carry any PCB.

The format of the 1xRTT Finger Task Instruction is described below:

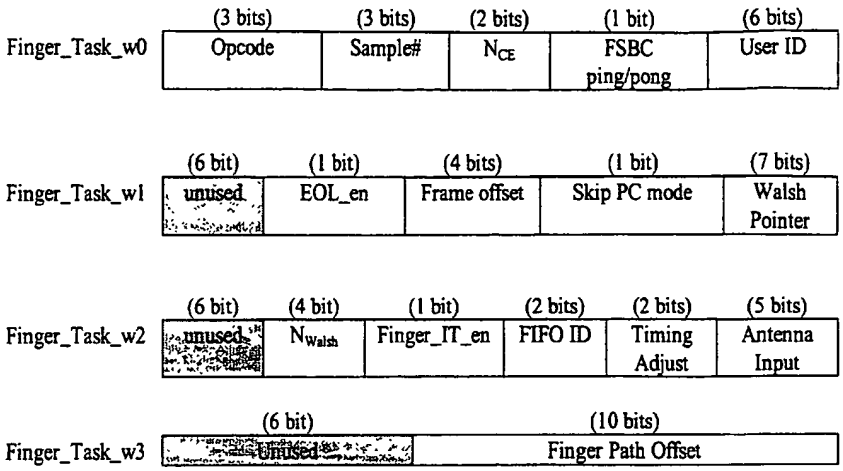


Figure 6-23: 1xRTT Finger Task Instruction Description

The 1xRTT finger task includes the following parameters:

- Task opcode: xxxx (*TBD*)
- Sample#: selects which sub-chips samples to process out of the 8 sub-chips.
- Frame offset: this is a system parameter that specifies a time skewing of Data Channel frames from System Time defined in 1xRTT/IS-95 standards in integer multiple of 1 PCG=1.25ms. The offset is specified in number of PCG.
- UserID: gives the Long Code Channel Mask.
- Walsh Pointer: specifies a Walsh sub-task set[10].
- N_{Walsh}: the number of active Walsh sub-tasks minus one. The Walsh sub-block contains 8 entries, one per Walsh sub-task. N_{Walsh} specifies the number of Walsh sub-tasks.
- EOL_en: this parameter specifies if one of the Walsh sub-tasks is an EOL Walsh sub-task. If "1", the first Walsh sub-task must be the EOL Walsh sub-task.

[10] If the information contained in the a Walsh blocks needs to be changed (e.g. Adding a new Walsh code) while the finger task is running, we just set up an unused Walsh Block and then change the Walsh pointer in the Finger task instruction.

- Finger_IT_en: finger interrupt enabled. If "1", Finger task generates an interrupt at the end of every PCG.
- FIFO ID: specifies which interrupt FIFO is used.
- FSBC ping/pong: a "0" indicates to use the ping FSB configuration buffer to determine where to ouput symbols, a "1" indicates to use the pong FSB configuration buffer.
- Antenna Input: specifies which antenna input buffer to read I&Q data from.
- Timing_Adjust: these bits reflect a change in sampling time for the Finger task. The options are no change, +/- delta, where delta is 1 sub-sample of a chip. The timing update is not reflected in the sample field of the task; the information is stored internally to the CCP. The following values are used:
  - "00": no change
  - "01": +1 chip sample adjustment (to later sample)
  - "1X": -1 chip sample adjustment (to earlier sample)

  To correctly adjust the Finger tasks timing, the Task Timing Adjust Request bit must be set for the task, as well as any changes to the Finger task's sampling time and long-code offset fields of the task description written to the Task Buffer and swapped-in. The Timing Adjust Request informs the CCP of the changes to sampling time and long-code offset of the Finger task that have just happened so that it could modify its internal processing accordingly. If changes to either the sampling time or long-code offset are made, and a Task Timing Adjust Request not made, the Finger task may output erroneous results and trigger interrupts at incorrect times.
- Skip_PC: specifies if the coherent accumulations on the pilot channel during EOL processing have to take into account the PC bit or not. If Skip_PC is set to "1", the coherent accumulation on the pilot channel is performed on the pilot symbols only. The pilot PC bit location in the PCG is hardwired accordingly to the 1xRTT standard. If EOL is specified on data, this parameter is not significant.
- N_{CE}: defines the number of chips coherently accumulated on the pilot channel to compute Raw Pilot (sub-) Symbols that are output in the RPB.
- Finger Path Offset: each Finger task corresponds to a defined path with an associated delay. This offset (or delay) is described with respect to System Time in number of chips. Each individual Finger task has an associated path-offset register.

Cycles per CCP iteration:
This task requires a number of cycles equal to the number of non-EOL sub-tasks plus the number of cycles required for an EOL Walsh sub-task (only one allowed per Finger task). An EOL Walsh sub-task takes three cycles.

6.2.2 Walsh Sub-Task

6.2.2.1 Walsh Sub-Task

The Walsh sub-task format is used to specify the entries in the Walsh Table. Entries in the Walsh Table are used for Finger tasks. If there is an EOL Walsh sub-task, it must be at the first location.

If less than the maximum number of entries (8 or 4) is needed, there are two options:
1) the Walsh sub-tasks must appear at the beginning of the Walsh set and be consecutive, the N_{walsh} field is set to the appropriate size; or
2) Any Walsh sub-task, which is not used in the first N_{walsh}, must have its Walsh enable bit cleared to indicate that it is disabled. The drawback to the second method is that a cycle is expended for each of the N_{walsh} even those sub-tasks that are disabled.

Walsh sub-tasks are operated in consecutive address order within a set.

6.2.2.2 Walsh sub-tasks format for Finger tasks (non-EOL):

When a Finger task (non-EOL) is running, the Walsh sub-task entries specify various parameters used for a particular UserID/WalshID combination.

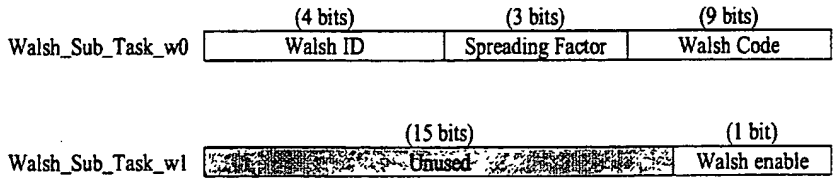


Figure 6-24: non-EOL Walsh sub-task Instruction Description

- Walsh enable: specifies if this Walsh sub-task is enabled or disabled. "1" means enables.

- Walsh ID: a user defined field that is unique in a particular Walsh sub-task set of the Walsh Table. It may be re-used in other Walsh sub-tasks sets in the Walsh Table. It is used (in combination to the Finger ID) to determine where in the Finger Symbol Buffer are stored de-spread symbols.
- Walsh Code: specifies the Walsh Hadamard code number.
- Spreading Factor (SF): specifies which spreading factor is used. (The spreading factor may vary from 4 up to 512).

6.2.2.3 Walsh sub-tasks format for Finger EOL tasks:

When a Finger EOL task is running, the Walsh sub-task entries specify various parameters used for a particular UserID/WalshID combination.

- Data Mode: Finger EOL task is running any Reverse Data Channel
- Pilot Mode: Finger EOL task is running on the Reverse Control Channel. Two modes are available (Mode0/ Mode1)
  - Mode0: in that mode, coherent accumulations are performed over PS_SIZE/N_{CA} chips. Power Control Bit of the Power Control Group (*PCG*) is always accumulated,
  - Mode1: in that mode, coherent accumulations are performed over (N_{CA}*SLOT_SIZE) chips. Power Control Bit of each Power Control Group is always skipped. The pilot PC bit location in the PCG is hardwired accordingly to the 1xRTT standard.

The Walsh sub-task parameters for the Finger EOL tasks are the following:

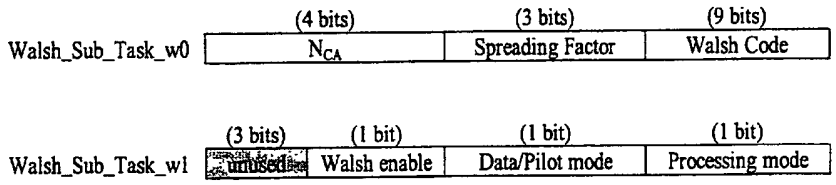


Figure 6-25: Walsh sub-task Instruction for Finger EOL tasks.

- Walsh enable. Specifies if this Walsh sub-task is enable or disable. "1" means enable.
- Walsh code: specifies the Walsh-Hadamard code number. 0 for the preamble R-ACH.
- Processing mode: specifies in which mode (Mode0/Mode1), the Finger EOL task is operating.
- N_{CA}: if Mode0, N_{CA} specifies the fraction of PS_SIZE on which coherent accumulations are performed (*resulting in coherent packets*). If Mode1, N_{CA} is the number of slots on which coherent accumulations are performed.
- Data/Pilot Mode: specifies whether we use the Reverse Control Channel (Data/Pilot Mode bit set to "0") or any Reverse Data channel (bit set to "1") to perform the Finger EOL.
- SF: allows selecting a Spreading Factor from 4 up to 512. (Spreading Factor = 2^{SF+2})

6.2.3 IS-95 Finger task

The IS-95 finger task is used for despreading IS-95 fingers while performing EOL measurement (DLL support). The despread symbols form consecutive 64 length Walsh sequences which are demodulated by a Fast Hadamard Transform outside of the CCP.

The finger task can generate one interruption every SLOT_SIZE*PS_SIZE chips.

The IS-95 Reverse Channel is single channel per user and thus, do not implement additional Walsh encryption on the Long Code + Short Code spreading to separate various channels of a given user. Consequently, unlike 1xRTT tasks, IS-95 Finger and DPE/Search tasks do not specify any Walsh Code.

EOL measurements are performed on the same (and unique) channel used to despread the symbols. Chips are integrated coherently over one symbol duration, then symbol energies are continuously dumped. Symbol duration is defined by the constant and hardwired PN Chips/ Walsh Chips SF of the IS-95 Reverse Traffic/Access Channel: 4.

Energy results are output once per frame in the EOL Buffer, updating the previous values.

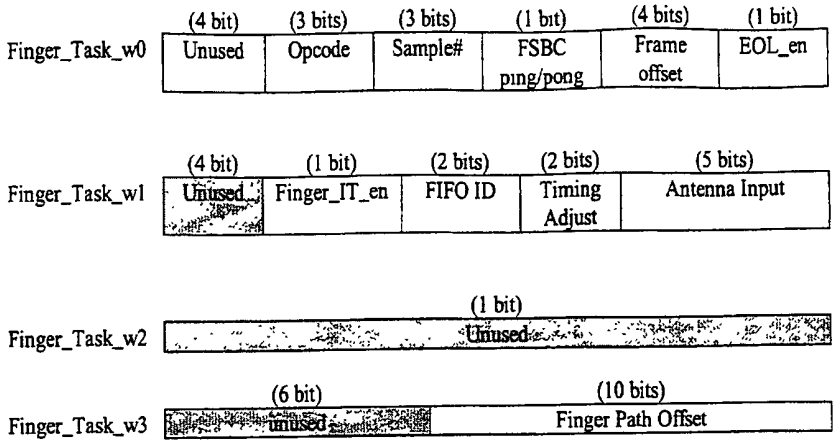


Figure 6-26: IS-95 Finger Task Instruction Description

The IS-95 finger task includes the following parameters:

- Task opcode: xxxx (*TBD*)
- Sample#: selects which sub-chips samples to process out of the 8 sub-chips.
- Frame offset: this is a system parameter that specifies a time skewing of Data Channel frames from System Time defined in 1xRTT/IS-95 standards in integer multiple of 1 slot defined by SLOT_SIZE. The offset is specified in number of slots.
- EOL_en: If "1", the EOL process is enabled.
- Finger_IT_en: finger interrupt enabled. If "1", Finger task generates an interrupt at the end of every slot defined by SLOT_SIZE chips.
- FIFO ID: specifies which interrupt FIFO is used.
- FSBC ping/pong: a "0" indicates to use the ping FSB configuration buffer to determine where to output symbols, a "1" indicates to use the pong FSB configuration buffer.
- Antenna Input: specifies which antenna input buffer to read I&Q data from.
- Timing_Adjust: these bits reflect a change in sampling time for the Finger task. The options are no change, +/- delta, where delta is 1 sub-sample of a chip. The timing update is not reflected in the sample field of the task; the information is stored internally to the ECP. The following values are used:

- "00": no change
- "01": +1 chip sample adjustment (to later sample)
- "1X": -1 chip sample adjustment (to earlier sample)

To correctly adjust the Finger tasks timing, the Task Timing Adjust Request bit must be set for the task, as well as any changes to the Finger task's sampling time and long-code offset fields of the task description written to the Task Buffer and swapped-in. The Timing Adjust Request informs the CCP of the changes to sampling time and long-code offset of the Finger task that have just happened so that it could modify its internal processing accordingly. If changes to either the sampling time or long-code offset are made, and a Task Timing Adjust Request not made, the Finger task may output erroneous results and trigger interrupts at incorrect times.

- Finger Path Offset: each Finger task corresponds to a defined path with an associated delay. This offset (or delay) is described with respect to System Time in number of chips. Each individual Finger task has an associated path-offset register.
- Walsh ID: a user defined field that is unique in a particular Walsh sub-task set of the Walsh Table. It may be re-used in other Walsh sub-tasks sets in the Walsh Table. It is used (in combination to the Finger ID) to determine where in the Finger Symbol Buffer are stored de-spread symbols.

Cycles per CCP iteration:
This task requires 1 cycle without EOL and 4 cycles with EOL.

7. Search Data Path Control

7.1 Software Interface

This section describes all registers and all configuration tables that the host processor has to access in order to either configure the Search Data Path or get the status information.

7.1.1 Search Task Buffer

The Search Task Buffer is a collection of MAX_DPE ping/pong buffers each of which may contain a CCP task of 90 bits. The status of each of MAX_DPW tasks as to whether the ping or pong is available to the CCP for execution, and the other available for SW loading/modification, can be controlled on an individual task basis by SW.

Task formats are found in Section **Error! Reference source not found.**. Host processor may read entries modifiable by the SW.

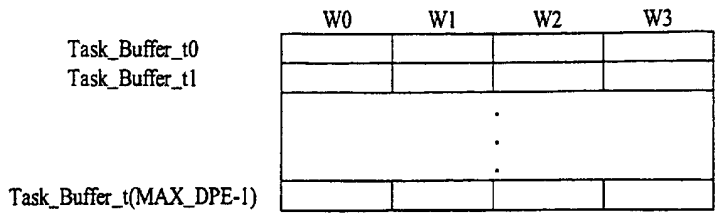


Figure 7-1: Task Buffer

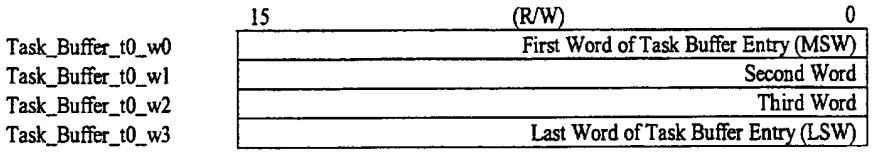


Figure 7-2: Task Buffer Entry Description

7.1.2 Synchronously Updated Configuration Parameters

These configuration parameters are double buffered and are updated at the Task-Update boundary: the software-modified register is copied to the hardware-readable shadow register at this time.

7.1.2.1 Search Task Request Bits

Search Task Requests to Start/Stop, Load/Reload, or Adjust Timing may be made at the next Task-Update boundary (following a write access) by setting the appropriate Task Request Bits. A "1" indicates a new request, '0' indicates no new request (requests may still be pending). The requesting will continue at succeeding Task-Update boundaries until the request is cleared by SW, via a "0", therefore, for proper operation, a "0" should be written at the Task-Update boundary immediately following a "1" being written. Refer to Figure 7-3. An operational description of Task Request is found in Section 5.1.2.

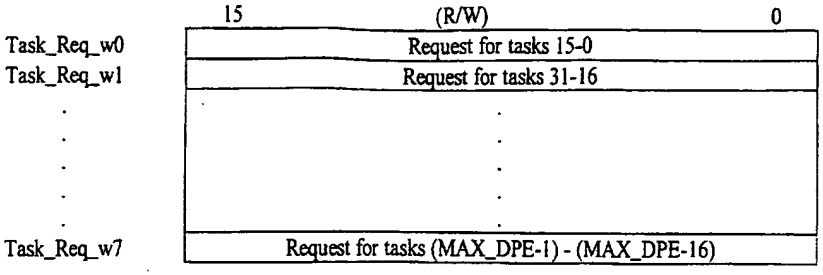


Figure 7-3: Search Task Buffer Entry Description

Writing the appropriate information to the task's Task Request ID field indicates the type of request.

7.1.2.2 Search Task RequestIDs

Each task has a Task Request ID register which identifies the type of Request being made when a "1" is written to the corresponding Task Request Bit. The format of the Task Request ID is:

	9	8 7	6	5	4	3 0
Task Request ID	Start/Stop Enable	Start/Stop ID	Load/Reload Enable	Load/ Reload ID	Timing Adjust Enable	Slot#
			R/W			

Figure 7-4: Search Task Request ID Register Format

- **Start/Stop Enable**: a "1" indicates that the request includes the start/stop action identified in the Start/Stop ID bits, a "0" indicates no start/stop action is requested.

- **Start/Stop ID**: These bits are referenced by the hardware only if the Start/Stop Enable bit is set ("1"). They have the following meaning:
  - "00" indicates Immediate Start
  - "01" indicates Synchronous Start
  - "10" indicates Immediate Stop
  - "11" indicates Synchronous Stop

- **Load/Reload Enable**: a "1" indicates that the request includes a load/reload action as identified in the Load/Reload ID bit, a "0" indicates no load/reload action is requested.

- **Load/Reload ID**: a "0" indicates an Immediate Load is requested, a "1" indicates a Synchronous Reload is requested. This bit is referenced by the hardware only if the

Load/Reload Enable bit is set ("1"). A synchronous Reload implies the instruction is "enabled", if it is not, the Reload will remain pending until an Immediate Load is requested.

- **Timing Ajust Enable** (for Finger Tasks only): a "1" is a request to adjust timing, and "0" is not a request to adjust timing. The timing change information is contained within the Finger Task description.

- **Slot #**: This field identifies the slot number for a synchronous action (synchronous start, synchronous stop, or synchronous reload). When a synchronous start is requested, this field defines the first slot to begin processing. When a synchronous stop is requested, this field defines the first slot which is not processed, in other words, the last slot number processed is the one, which precedes this slot number (modulo 16). When a synchronous reload is requested, this field defines the first slot after the reload, in other words, the last slot number processed before the reload is the one which precedes this slot number (modulo 16). The sixteen slots per frame are numbered 0 through 15.

7.1.3 Asynchronously Updated Configuration Parameters

Asynchronously updated configuration parameters are typically ones that are not changed often, if at all, once the CCP begins its operations. Changing these parameters when not in use is not a problem.

7.1.3.1 Search External Interrupt Enable Register

The Search External Interrupt Control register controls the enabling of FIFO, error and system interrupts to the external interrupt lines.

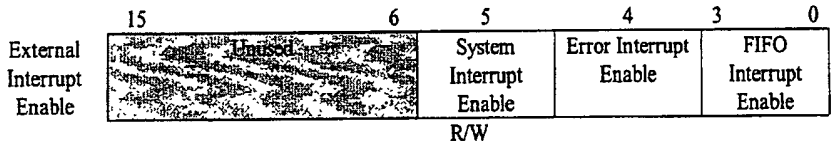


Figure 7-5: Search External Interrupt Enable Register Format

7.1.4 Search Data Path Status

7.1.4.1 Task Ping/Pong Status Bits

The host processor can read the current ping/pong status for all tasks in Task Buffer. "0" indicates that HW is reading ping and SW reading/modifying pong, "1" indicates the reverse.

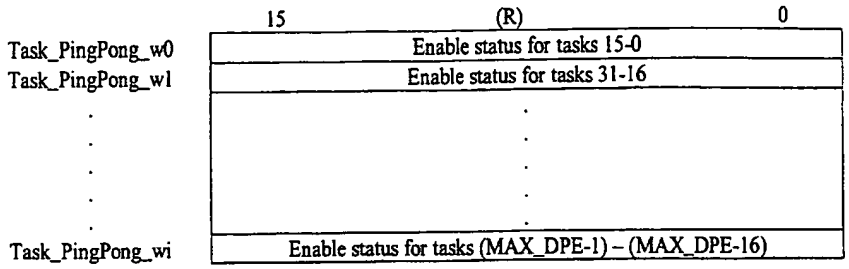


Figure 7-6: Task Ping/Pong status Bits

7.1.4.2 Task Run/Stop Status Bits

The host processor can read the current Run/Stop status for all tasks in Task Buffer.

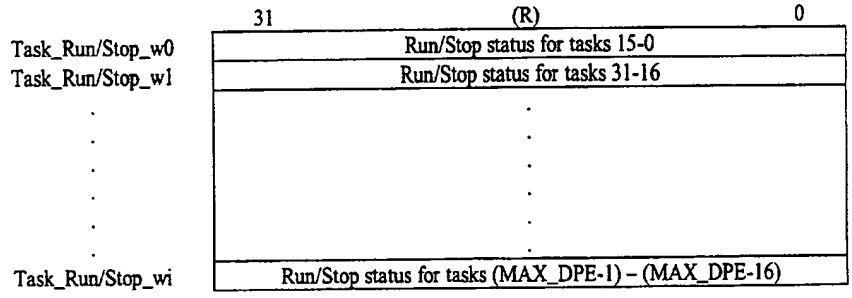


Figure 7-7: Task Run/Stop Status Bits

Each task has a two bit field which encodes the Run/Stop status in the following manner:

- "00" indicates "Stopped"
- "01" indicates "Waiting to Run"
- "10" indicates "Running"
- "11" indicates "Waiting to Stop"

7.1.4.3 Cycle_count Register

The Cycle_Count register captures the number of cycles expended in the most recent CCP iteration of the Rake Data Path; it is updated every CCP iteration.

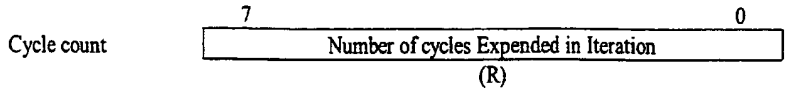


Figure 7-8: Cycle count register

7.1.5 Interrupt Status

A synchronizer circuit allows the processor to read accurate values from these registers.

7.1.5.1 Search_Int_Error_Event_Status Register

Reading the Search_Int_Error_Event_Status clears the Error interrupt event.

- Cycles Exceeded Error. 1 bit. "1" if the number of cycles attempted in an iteration was greater than the maximum (PAR_SEARCH*MAX_CYCLES), "0" otherwise.
- Interrupt FIFO Overflow Error. 4bits. One bit for each of the four interrupt FIFO's, "1" if the corresponding FIFO has overflowed, "0" otherwise.

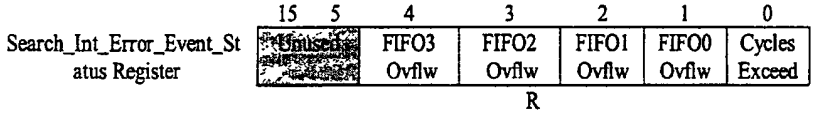


Figure 7-9: Search_Int_Error_Event_Status Register

7.1.5.2 Search FIFO Status & Content

The Search FIFO Status register shows each interrupt FIFO's empty/non-empty status. When a particular FIFO is not empty, its FIFO Empty Status is "1". The status bit is cleared when all FIFO contents have been read out.

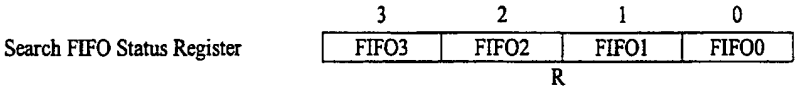


Figure 7-10: Search FIFO Status Register

Search FIFO contents are 2 words each, and are read out at one memory location, one after the other. When the second word is read out, the FIFO hardware increments its internal read pointer.

Figure 7-11: Search FIFO Content Format

The value of GCC2 is recorded in GCC2 field when the event occurred.

7.1.6 Output Data

7.1.6.1 DPE Buffer

The DPE Buffer is partitioned into equal-sized blocks, each with PAR_SEARCH*16 locations, a location holding an on-time energy and (optionally) a late-time energy for an offset within the search window.

Output results from the DPE tasks are placed in the DPE Buffer, which holds up to MAX_DPE*PAR_SEARCH*16*2 energy values.

A DPE task with a certain DPE Search ID outputs its results starting at the same block number in the DPE Buffer. For a DPE task, if the search window is less than or equal to PAR_SEARCH*16 chips, then all of its results are placed in one block of the DPE Buffer. If the search window is more than PAR_SEARCH*16, the results are placed in the next adjacent blocks of the DPE Buffer. Results are stored sequentially from the starting to the ending offset.

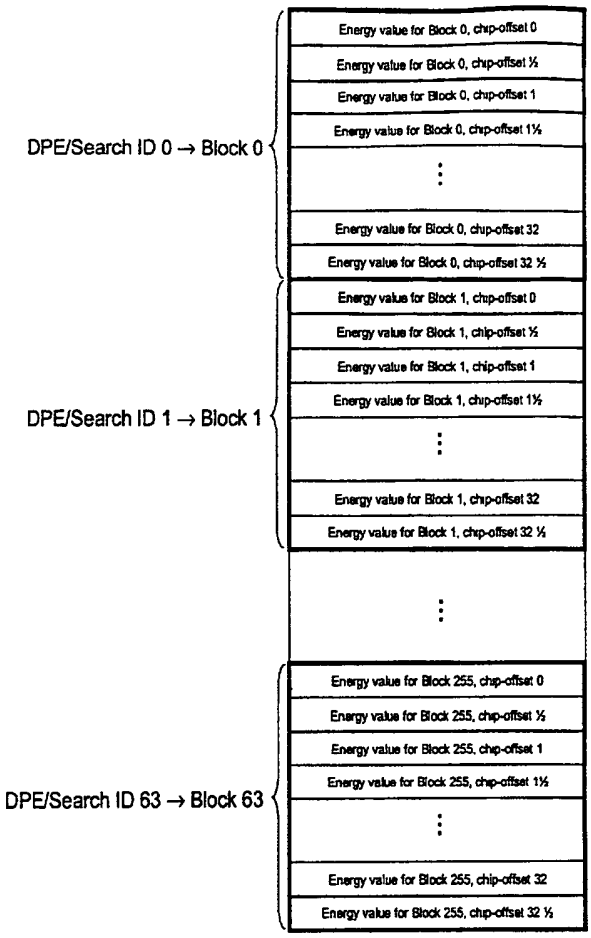


Figure 7-12: DPE Buffer

7.2 Search/DPE Task Description

The DPE Search task measures path energies starting at a specified start time and in a specified window of offset.

7.2.1 1xRTT DPE/Search task

The DPE programmability provides the same flexibility as the EOL one:

- Measurements may be done either on the Reverse Data Channel or on the Reverse Control Channel (Pilot/Data mode). (see Section **6.2.2.3**).
- Coherent integration length on pilot is wide range programmable using the N_{CA} parameter of the associated Walsh sub-task (see Section **6.2.2.3**).
- When processing on the pilot channel in coherent Mode 1 (see Section **6.2.2.3**), the PC bits can be skipped or not accordingly to the Skip_PC bit of the DPE task parameters.

- Data Mode: DPE task is running any Reverse Data Channel
- Pilot Mode: DPE task is running on the Reverse Control Channel. Two modes are available (Mode0/ Mode1)
  - Mode0: in that mode, coherent accumulations are performed over PS_SIZE/N_{CA} chips. Power Control Bit of the Power Control Group (*PCG*) is always accumulated,
  - Mode1: in that mode, coherent accumulations are performed over (N_{CA}*SLOT_SIZE) chips. Power Control Bit of each Power Control Group is always skipped. The pilot PC bit location in the PCG is hardwired accordingly to the 1xRTT standard.

Unlike the EOL which is continuously running, the DPE task is one shot and needs a "length" parameter. This parameter is provided in the associated DPE Walsh sub-task with N_{NTS} which specifies the total number of coherent packets to accumulate non-coherently.

The format of the 1xRTT DPE/Search task is described below:

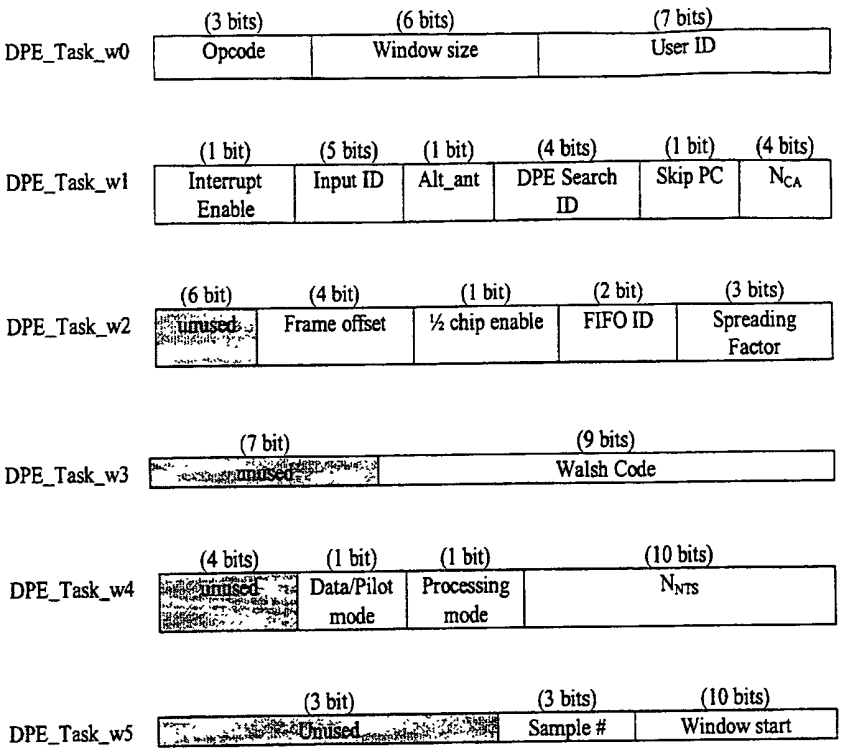


Figure 7-13: DPE/Search 1xRTT task Instruction description

The 1xRTT DPE/Search task includes the following parameters:

- Opcode: xxx. (*TBD*)
- Input ID: selects which input buffer (*antenna / sector selection*).
- Alt_ant: a "1" selects Alternating Antenna Mode from two input buffers: 00 and 01.
- DPE search ID: distinguishes different DPE tasks.
- Interrupt Enable: enables an interrupt sent when the DPE task is completed.
- FIFO ID: specifies which interrupt FIFO is used.
- User ID: gives the Long Code Channel Mask.

- Frame offset: this is a system parameter that specifies a time skewing of Data Channel frames from System Time defined in 1xRTT/IS-95 standards in integer multiple of 1 PCG=1.25ms. The offset is specified in number of PCG.
- ½ chip Enable: enables the processing of samples at ½ chip resolution.
- Skip_PC: specifies if the coherent accumulations in Mode1 on the pilot channel during DPE processing have to take into account the PC bit or not. If Skip_PC is set to "1", the coherent accumulation on the pilot channel is performed on the pilot symbols only. The pilot PC bit location in the PCG is hardwired accordingly to the 1xRTT standard.
- Window size: select the length of the window. $0 \le n < (\text{PAR\_SEARCH} * 16)$, gives a window size of (PAR_SEARCH*16)(n+1) chips.
- Sample#: DPE/Search task has the option to select a number of samples out of 8 on which the task have to operate.
- Window-start: specifies in number of chips with respect to the Internal Time (GCC2). The maximum window size in chips is given by MAX_WIN_SIZE. Of course, in that case, the Search Datapath can handle only one DPE task at a time.
- Walsh code: specifies the Walsh-Hadamard code number.
- N_{NTS}: specifies the non coherent accumulation length in number of:
  - Coherent packets in Pilot Mode
  - Slots in Data Mode.

  N_{NTS} actually specifies the length (*duration*) of the DPE task.
- Processing mode: specifies in which mode (Mode0/Mode1), the DPE task is operating.
- N_{CA}: if Mode0, N_{CA} specifies the fraction of PS_SIZE on which coherent accumulations are performed (*resulting in coherent packets*). If Mode1, N_{CA} is the number of slots on which coherent accumulations are performed.
- Data/Pilot Mode: specifies whether we use the Reverse Control Channel (Data/Pilot Mode bit set to "0") or any Reverse Data channel (bit set to "1") to perform the DPE.
- SF: allows selecting a Spreading Factor from 4 up to 512. (Spreading Factor = 2^{SF+2}) . (pertains only when operating in data mode).

Cycles per CCP iteration:

This task requires a number of cycles equal to (1/PAR_SEARCH) the number of offsets in the window size, regardless of whether ½-chip processing is enabled.

Activation Time:

This task is activated when the GCC2 modulo 98304 is equal to the value specified in the window start offset field.

7.2.2 IS-95 DPE/Search task

The DPE integration process is similar to the EOL one provided with the IS-95 Finger task:

Chips are integrated coherently over one symbol duration, then symbol energies are continuously dumped. Symbol duration is defined by the constant and hardwired PN Chips / Walsh Chips SF of the IS-95 Reverse Traffic/Access Channel: 4.

Unlike the EOL which is continuously running, the DPE task is one shot and needs a "length" parameter. This parameter is N_{NTS} which specifies the total number of coherent packets to accumulate non-coherently.

The IS-95 Reverse Channel is single channel per user and thus, do not implement additional Walsh encryption on the Long Code + Short Code spreading to separate various channels of a given user. Consequently, unlike 1xRTT tasks, IS-95 Finger and DPE/Search tasks do not specify any Walsh Code.

The format of the IS-95 DPE/Search task is described below:

Figure 7-14: DPE/Search IS-95 task Instruction description

The IS-95 DPE/Search task includes the following parameters:

- Opcode: xxx. (*TBD*)
- Input ID: selects which input buffer (*antenna / sector selection*).
- Alt_ant: a "1" selects Alternating Antenna Mode from two input buffers: 00 and 01.
- DPE search ID: distinguishes different DPE tasks.
- Interrupt Enable: enables an interrupt sent when the DPE task is completed.
- FIFO ID: specifies which interrupt FIFO is used.
- User ID: gives the Long Code Channel Mask.

- Frame offset: this is a system parameter that specifies a time skewing of Data Channel frames from System Time defined in 1xRTT/IS-95 standards in integer multiple of 1 slot defined by SLOT_SIZE. The offset is specified in number of slots.
- ½ chip Enable: enables the processing of samples at ½ chip resolution.
- Window size: select the length of the window. $0 \le n < (\text{PAR\_SEARCH} * 16)$, gives a window size of (PAR_SEARCH*16)(n+1) chips.
- Sample#: DPE/Search task has the option to select a number of samples out of 8 on which the task have to operate.
- Window-start: specifies in number of chips with respect to the Internal Time (GCC2). The maximum window size in chips is given by 4*MAX_CYCLES. Of course, in that case, the Search Datapath can handle only one DPE task at a time.
- N_{NTS}: coherent symbols are non-coherently accumulated over N_{NTS} slots. 1 slot is defined by SLOT_SIZE. N_{NTS} actually specifies the length (*duration*) of the DPE task.

Cycles per CCP iteration:
This task requires a number of cycle equal to (1/PAR_SEARCH) the number of offsets in the window size, regardless of whether ½-chip processing is enabled.

Activation Time:
This task is activated when the GCC2 modulo 98304 is equal to the value specified in the window start offset field.

8. External Interface

<TBD>

9. References

[1] Motorola Plus Encore Receiver board. http://www.mot.com/ies/GPS/pdfs/ut.pdf.
[2] The CDMA2000 ITU-R RTT Candidate Submission.
[3] Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System. TIA/EIA/IS-95-A. May 1995.
[4] Base Station synchronization to GPS in 1xRTT/IS-95. Ver: 1.0. TI internal. December 1999.

Correlator Co-Processor

Specification

Ver: 0.6

File: www.asic.sc.ti.com/~yklee/documents/ccp_main.pdf

Department: Application Specific Product / Wireless Communication Systems

Contact: Yuan Kang Lee, yklee@ti.com

Contributors

Authors
Kathy Brown
Frank Honore
Yuan Kang Lee
S. Sriram

History

Version	Date	Notes
Ver: 0.2	20-December-98	1
Ver: 0.2x	21-January-99	2
Ver: 0.2a	24-January-99	3
Ver: 0.3x	16-April-99	4
Ver: 0.6	16-August-99	5
Ver: 0.5	15-October-99	6
Ver: 0.6	16-November-99	7

Notes:

1. First release.
2. Unofficial release
3. Revision of version 0.2x
4. Version for review
5. Updates before specification freeze.
6. Changes for first prototype.
7. Additions for compressed and sleep modes.

REVIEW LIST

Reviewer	Approval Date
Z. Gu	
D. Hutchison	
Y. K. Lee	
Partha Mukherjee	
Chaitali Sengupta	
C. Tracy	
Y. Wong	

Table of Contents

Figure Summary

Table Summary

1. Introduction

The Correlator Co-Processor (CCP) is a programmable, highly flexible, vector-based correlation machine that performs CDMA mobile-station RAKE receiver operations. Because most RAKE receiver functions involve correlations and accumulations, regardless of the particular wireless protocol, a centralized correlation machine (such as the CCP) can be used for various RAKE receiver tasks like finger despreading and search. Each RAKE receiver task uses the same centralized datapath of the CCP in a time-multiplexed fashion, so that many different tasks can be performed on the CCP "simultaneously." To reduce power dissipation, the main datapath is vectorized.

The sharing of the main CCP datapath by RAKE receiver functions (i.e., finger demodulation, code tracking loops, and search) means that the amount of CCP's resources used for each of the functions is not "hardwired" and is configurable. The amount of resources can be allocated in a way desired by the user and as befitting the situation. Unused resources are turned off and add minimally to the total power dissipated. Section 1.5 describes how the CCP's resources are divided up for RAKE receiver functions and shows examples on the number of fingers, number of code channels, number of search tasks, search window sizes, etc. that can be supported by the CCP.

In addition to performing correlations (complex valued), which consist of despreading and coherent accumulation, the CCP also accumulates "symbol" energy values (called non-coherent accumulations). For example, it accumulates the early, on-time, and late samples of a RAKE finger; these measurements are used for the finger's code-tracking loop (typically a delay-lock loop or DLL). For search operations, the CCP returns the accumulated energy values for a specified window of offsets.

The CCP does not perform "symbol"-rate receiver operations such as channel estimation, maximal-ratio combining, and de-interleaving, nor feedback loops such as AGC, AFC, and DLL. (For DLL, the CCP supplies the energy values to the feedback loop, but it does not operate on the loop itself.) These symbol operations can be performed either in a DSP or in dedicated hardware outside of the CCP.

Many versions of the CCP may be implemented using this specification. For this reason, some parameters will be referred to with names rather than numbers (e.g. SYS_CLK, MAX_CYCLES). The paragraph in the implementation section which refers to a particular version will define these parameters.

1.1 Implementation Parameters

1.1.1 LoneStar ASIC Version

Parameter Values:

- SYS_CLK = 16X (61.44 MHz)
- MAX_CYCLES = 256
- MAX_TASKS = 64
- MAX_FINGERS = 64
- TOTAL_WALSH = 32
- NUM_LARGE_FINGERS = 16
- NUM_WALSH8 = 16
- NUM_WALSH4 = 16
- TEMP_BUFFER_SIZE = 16
- MAX_LCI = 8
- MAX_DPE = 0

Tasks not implemented:

- DPE
- PSC Search

1.1.2 LoneStar FPGA Version

Parameter Values:

- SYS_CLK = 4X (15.36 MHz)
- MAX_CYCLES = 64
- MAX_TASKS = 16

- MAX_FINGERS = 16
- TOTAL_WALSH = 8
- NUM_LARGE_FINGERS = 0
- NUM_WALSH8 = 0
- NUM_WALSH4 = 8
- TEMP_BUFFER_SIZE = 8
- MAX_LCI = 8
- MAX_DPE = 0

Tasks not implemented:

- DPE
- PSC Search

Features not supported:

- Compressed Mode
- Sleep Mode

Other changes:

EOL sub-task of Finger task computes and stores only early and late results.

1.2 Wireless Protocol Support

This version of the CCP specification supports the IMT2000-DS, 3.84 MHz chip rate specifiation. With modifications, it could support various other spread-spectrum, CDMA communications systems such as:

- IMT2000-MC
- IMT2000-TDD
- CDMAOne/IS-95
- GPS

1.3 System Requirements

1.3.1 Input Clocks

Let 1X be a clock rate of 3.84 MHz, i.e., the baseline IMT2000 chip rate. The CCP requires the following input clocks:

- System clock: N times 4X clock (SYS_CLK)
- Receive sample clock: 8X clock (30.72 MHz) or 4X (15.36 MHz) or 2X(7.68 MHz).

These two clocks must be synchronous, i.e., phase aligned.

Using a 4X clock reduces CCP input buffer requirements and A/D power at the cost of reduced signal-to-noise ratio (SNR). A 2X clock might be used in a CCP which implemented only search tasks. The receive sample clock only affects the Input Buffer and not the core of CCP. Refer to the top-level block diagram of the CCP in Figure 2.

1.3.2 Receiver I/Q Samples

In-phase (I) and quadrature (Q) samples are up to 6 bits each. The CCP may receive multiple I/Q samples from multiple sources for antenna diversity. The buffers that store the I/Q samples are outside of the core of CCP. Four sources may be supported. Each source may be 2 or 4 or 8 times the chip rate.

1.3.3 System Bus

The CCP is a RHEA peripheral. Full functionality of the CCP can be realized by using only the RHEA bus, subject to the limitations of the bandwidth of the RHEA bus.

For flexibility in retrieving finger data, the Finger Symbol Buffer (which stores finger data) can also be accessed via the FSB External Bus. An external agent can retrieve this data, for example, for hardware-based symbol processing. The usage of this bus depends on how the CCP is connected in the DBB system.

1.4 Features

The CCP performs correlations and accumulations (coherent and non-coherent). The basic operation is the correlation, commonly referred to as de-spreading (followed by sum-and-dump), which is used to remove the effects of long-code scrambling and Walsh spreading (using WCDMA nomenclature). The CCP controls correlations and accumulations in certain ways to achieve all RAKE and searcher operations.

The CCP is a multi-tasking machine. For example, a CCP finger operation is called a Finger task. The CCP tasks are:

- Finger task: Walsh despreading of one or more channels, and early-on-time-late (EOL) energy measurement
- Delay Profile Estimation (DPE) task or "path search"
- Primary Search Code (PSC) Search task or "stage 1 search"
- Secondary Search Code (SSC) Search or "stage 2 search"
- Long Code Identifier (LCI) Search task or "stage 3 search"

These tasks and their programmable features are briefly described in the sections that follow. Exact details on all tasks are found in Section 6.

Not all tasks may be implemented in a given version of the CCP, tasks which are not implemented result in a no-op consuming 1 cycle if executed.

1.4.1 Finger

In a CCP Finger task, long-code (LC) de-scrambling and Walsh de-spreading are performed, resulting in a complex symbol stream (I and Q) that is output. Each finger operation can accommodate multiple Walsh code channels, but all channels must have the same LC offset, frame offset, slot offset, and spreading factor.

To aid in the combining of symbols, a maximum energy for each Walsh channel of each Finger is output once per slot.

Each finger operation supports one set of early-ontime-late (EOL) energy measurements that is made on a particular Walsh code channel. The energy measurements are output once per frame.

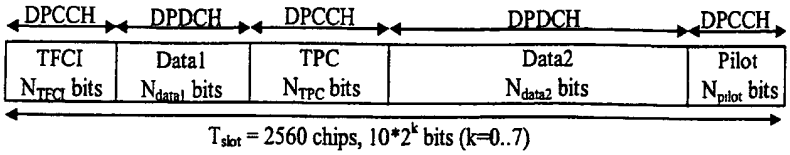


Figure 1. Programmable Pilot Position and Size

Because the Pilot and Transmit-Power-Control (TPC) bit(s) locations may differ between the different protocol standards, they can be configured in the CCP. Figure 1 shows an IMT2000-DS example of the Downlink Dedicated Physical Channel (DPCH) which has the pilot bits at the end of the slot, and the TPC bit(s) in the middle.

To support code-tracking loop operations, or DLL, each CCP finger supports one set of Early/On-time/Late (EOL) energy measurements. Open-loop (STTD) transmit diversity is supported.

There are two ways to specify the measurement of Pilot energy:

1. Each Pilot symbol is converted into an energy value, and the energy values of all Pilot symbols in one frame are accumulated.

2. The Pilot symbols are accumulated over one or multiple slots, and the resulting complex value is converted into one energy value. All such resulting energy values inside each frame are then accumulated together. For example, if Pilot symbols are accumulated over three (3) slots, the resulting 5 energy values (15 slots divided by 3) in the frame are summed together and output. For STTD, the energy of both antennas is combined.

There is one way to specify the measurement of non-Pilot energy:

1. Each symbol is converted into an energy value, and the energy values of all symbols in one frame are accumulated.

1.4.2 Delay Profile Estimation (DPE)

The Delay Profile Estimation (DPE) task is used to identify potential multi-paths in a window of offsets. An energy value is returned for every chip or ½-chip offset in the specified offset window. Measurements take place over a specified number of radio slots and are output periodically.

The DPE task can measure the energy exactly like the EOL measurements of the finger task. Refer to Section 1.4.1.

1.4.3 Primary Search Code (PSC) Search or "stage 1 search"

The Primary Search Code (PSC) Search task is used for acquisition of new base stations, also referred to as "Stage 1" search in the WCDMA standards. It returns the slot-periodic matched-filter output energy values (for 5120 ½-chip offsets). Measurements take place over a specified number of radio slots.

The CCP takes advantage of the hierarchical structure in the 256-chip PSC and performs the matched-filtering in two steps of 16 operations each, the first step consuming CCP cycles and the second step being run on post-processing hardware. The "16x16" PSC sequence is programmable.

1.4.4 Second Search Code (SSC) Search or "stage 2 search"

Second Search Code (SSC) Search, or "stage 2 search", is used to establish frame synchronization of a new base station and to identify its long-code group. The SSC Search task assumes the SSC Walsh codes to have 16x16 structure and "de-spreads" the data stream with a spreading factor of 16 within the 256 chip PSC/SSC search code position. In addition, the PSC symbols are de-spread at the same time. Hence, 16 complex symbols per slot are output for the SSC and 16 symbols per slot are output for the PSC. The the remaining operations of the Walsh-Hadamard transform must be done outside of the CCP, in a DSP or dedicated hardware

1.4.5 Long Code Identifier (LCI) Search or "stage 3 search"

Long Code Identifier (LCI) search, or "stage 3 search," is used to determine the exact scrambling code, following search stages 1 and 2. This CCP task measures the de-spread energies using each of the specified long scrambling codes.

The LCI search task can measure the Common Pilot energy exactly like the Pilot measurements of the DPE task. Refer to Section 1.4.1.

1.5 CCP resource allocation

In this section, we briefly describe the different ways that CCP resources can be divided to perform the RAKE receiver functions described in the previous sections.

There are a total of MAX_CYCLES "correlation cycles", or cycles, supported by the CCP. In other words, the CCP has a processing capacity roughly equivalent to MAX_CYCLES discrete Correlators for RAKE fingers. The CCP has 4*MAX_CYCLES equivalent Correlators for DPE search. The number of correlation cycles expended for each CCP task is as follows:

- Finger task. Each Walsh channel uses one correlation cycle, independent of its spreading factor. Each set of EOL measurements uses three correlation cycles.
- DPE task. Window sizes are a multiple of 16 chips. Every 2 chips of window size uses only one cycle, even though two equivalent correlations are performed, or four equivalent correlations performed for the on-time and on-time + ½-chip samples when the ½ chip option is enabled.
- PSC Search task. This task uses 16 cycles.
- SSC Search task. This task uses 2 cycles.
- LCI Search task. This task uses one cycle per long-code.

The CCP supports MAX_FINGERS simultaneous Finger tasks; NUM_WALSH8 of these Finger tasks support up to 8 Walsh channels, and NUM_WALSH4 support up to 4 Walsh channels. Each Finger task may support 1 set of EOL measurements, but a Finger task making EOL measurements will reduce by one the number of Walsh channels supported on that finger. Hence, Finger tasks may specify up to a maximum of (32 x (NUM_WALSH8 –1) + 32 x (NUM_WALSH4 -1) + 3 x MAX_FINGERS) cycles if necessary (only MAX_CYCLES cycles could be run at any one time).

The CCP supports one PSC Search task. If present, it must be the first task to run, in order for the post-processing of the results to have time to be completed.

The CCP supports up to MAX_DPE DPE Search tasks. This number can be supported only if the sum of the sizes of the search windows is less than MAX_CYCLES.

The CCP supports up to MAX_LCI LCI Search tasks.

The CCP supports a total of MAX_TASKS tasks in any combination that conforms to the rules above. All tasks together cannot exceed MAX_CYCLES cycles.

A disabled task does not consume any cycles. An enabled task which is not running comsumes one cycle per task. An enabled task which is not implemented consumes one cycle per task.

1.5.1 Examples of CCP Resource Allocation

Using the rules outlined in Section 1.5, the following shows sample scenarios supported by the CCP.

Scenario 1:
1 DPE task of 256-chip offsets (total cycles used = 128).

This scenario could occur during an inter-frequency base station measurement or upon "wake" in standby mode. Using full CCP capabilities is justified because quick measurement or re-acquisition is important.

Scenario 2:
2 DPE tasks of 256-chip offsets each (total cycles used = 256).

This scenario could occur during an inter-frequency base station measurement.

Scenario 3:
1 DPE task of 256-chip offset
1 PSC Search task
6 Finger tasks of 16 Walsh codes (implemented as 12 fingers of 8 codes each)
6 Finger tasks of EOL measurements
16 cycles reserved for transfers from temporary buffer to configuration memories

total cycles used = 256/2 + 16 + 6*16 + 6*3 + 16 = 274

This could be a steady-state scenario with one radio link that consists of 16 Walsh channels. The full usage of CCP resources is needed because of the large number of Walsh channels.

Scenario 4:
1 DPE task of 256-chip offset
1 PSC search task
6 Finger tasks of 4 Walsh codes
6 Finger tasks of 4 Walsh codes
6 Finger tasks of 2 Walsh codes
6 Finger tasks of 1 Walsh code
6 Finger tasks of EOL measurements

total cycles used = 256/2 + 16 + 6*4 + 6*4 + 6*2 + 6*1 + 6*3 = 228

This could be a steady-state scenario with four radio links, two with 4 codes, one with 2 codes, and one with 1 code.

Scenario 5:
1 DPE task of 128-chip offset
6 Finger tasks of 1 Walsh code, each with EOL measurements

total cycles used = 128/2 + 6*1 + 6*3 = 88

This could be a steady-state scenario with one radio link each with one code. The DPE search window is small because delay spread is small or because a large search is divided into parts.

Scenario 6:
1 DPE task of 128-chip offset
2 Finger tasks of 4 Walsh codes each, plus EOL measurements for each

total cycles used = 128/2 + 2*4 + 2*3 = 78

This could be a 2 Mbps scenario (indoor).

1.6 Digital Baseband System Partitioning

As discussed in Section 1.4, the CCP performs "chip"-rate processing and energy accumulation, and it does not perform "symbol"-rate receiver operations such as channel estimation, maximal-ratio combining, and de-interleaving, nor feedback loops such as AGC, AFC, and DLL. (For DLL, the CCP supplies the energy values to the feedback loop, but it does not operate on the loop itself.) These symbol operations can be performed either in a DSP or in dedicated hardware outside of the CCP.

1.7 Roadmap

This section discusses possible evolutions of the CCP's features and architecture.

1.7.1 WCDMA Chip Rate (Spreading Bandwidth)

The CCP currently supports the WCDMA chip rate of 3.84 MHz. Although the basic architecture concept of the CCP would remain the same for other chip rates, the CCP's capabilities and its architecture have been defined with the 3.84 chip rate in mind. For example, the number of needed fingers and multipath delay-spread are functions of the spreading bandwidth. Both these factors would affect the capabilities of the CCP and its architecture. These are just some of the many factors involved.

The architecture of the CCP also depends greatly on the ASIC technology that is available at the time when higher spreading bandwidths (7.68 and 15.36 MHz) are needed. Trade-off comparisons, such as datapath width versus clock rate, will be made for the targeted ASIC technology.

1.7.2 Time Division Duplex (TDD) Mode

Time Division Duplex (TDD) mode operation of the CCP is currently under study and will be supported by the CCP.

1.7.3 Digital Baseband Integration

Additional features are under study to enable the following:

- Receiver/transmitter synchronization
- Signal strength estimation
- Sleep-mode operations

1.7.4 Antenna Diversity

Additional features are under study to enable switched-antenna diversity.

1.7.5 IMT2000-MC/IS-95

IMT2000-MC and IS-95 operations are under study.

1.7.6 GPS

GPS operations are under study.

1.7.7 Post processing of search results

Currently, the CCP simply outputs raw energy values from PSC Search and DPE. Adding post-processing features such as peak finding is under study.

1.7.8 Inter-frequency Handover

Slotted downlink reception operations are under study. Some enabling features have been incorporated.

2. Architecture Overview

The top-level of the CCP is shown in Figure 2. The key idea in the CCP design is to buffer a block of receive (sub)chip-rate samples and perform all the correlations repeatedly using this data. Also, the CCP datapath performs a partial correlation using multiple chips of data in a single cycle. In the following sections, we briefly describe each of the major components in the CCP, as shown in Figure 2.

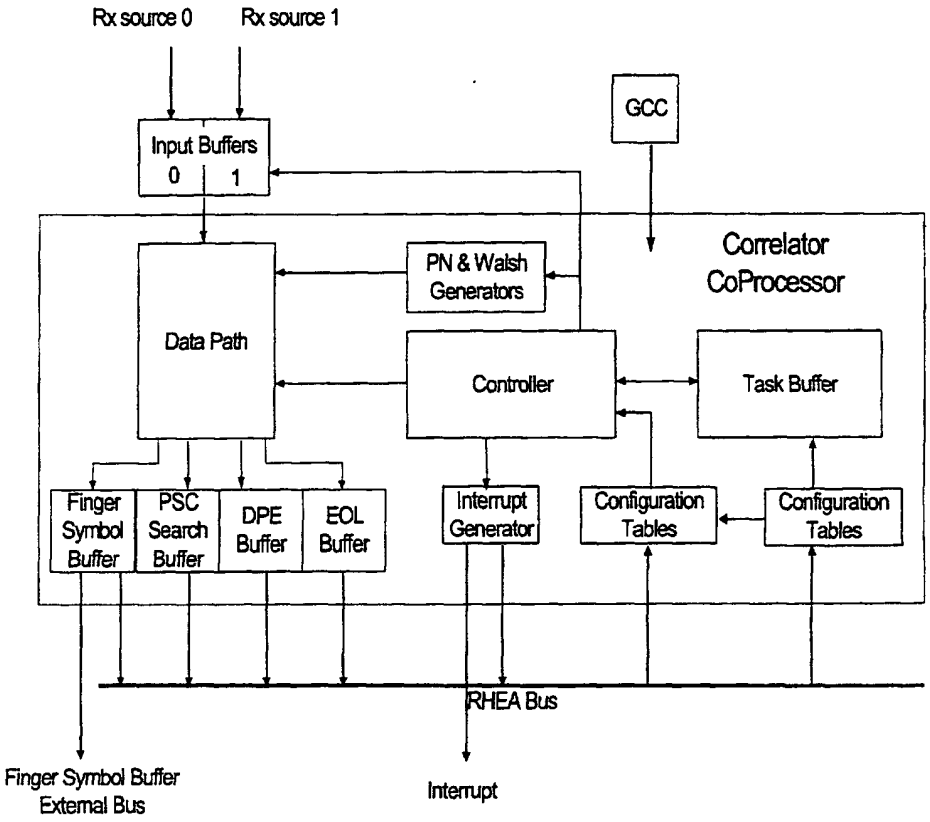


Figure 2. CCP Top-Level Block Diagram

2.1 Input Buffers

The Input Buffers store a stream of receive I/Q sub-chip samples for processing by the CCP Datapath. These sub-chip samples may come directly from an analog front-end (AFE), digital front-end filtering, or digital interpolation filtering. For each correlation cycle, the CCP selects a set of input samples corresponding to a particular sub-chip

sample. The CCP may receive input data from multiple sources, for example, to support multiple antennae. Figure 2 shows two sources. Each input source may be at a 4X or 8X chip rate. Up to four input sources are supported. The Input Buffers may be implemented as custom register files.

2.2 Datapath

The CCP's datapath is shown in Figure 3. It consists of "multipliers" to multiply the samples from the input buffer with samples of PN and Walsh codes, adder trees to generate partial correlations, a coherent accumulator to sum the current partial correlation with previous partial correlation(s), and a post processing block for performing RMS ($\text{sqrt}(I^2 + Q^2)$) calculations (sometimes referred to as "energy" values) and non-coherent accumulations.

The CCP datapath employs extensive pipeline stages to maximize computational capacity.

Temporary ("scratch") memories are used to store partial correlation and intermediate RMS accumulation results. Task results are stored in one of the output buffers -- Finger Symbol Buffer, DPE Buffer, LCI Buffer, EOL Buffer, SSC Search Buffer, or PSC Search Buffer -- depending on the type of that task.

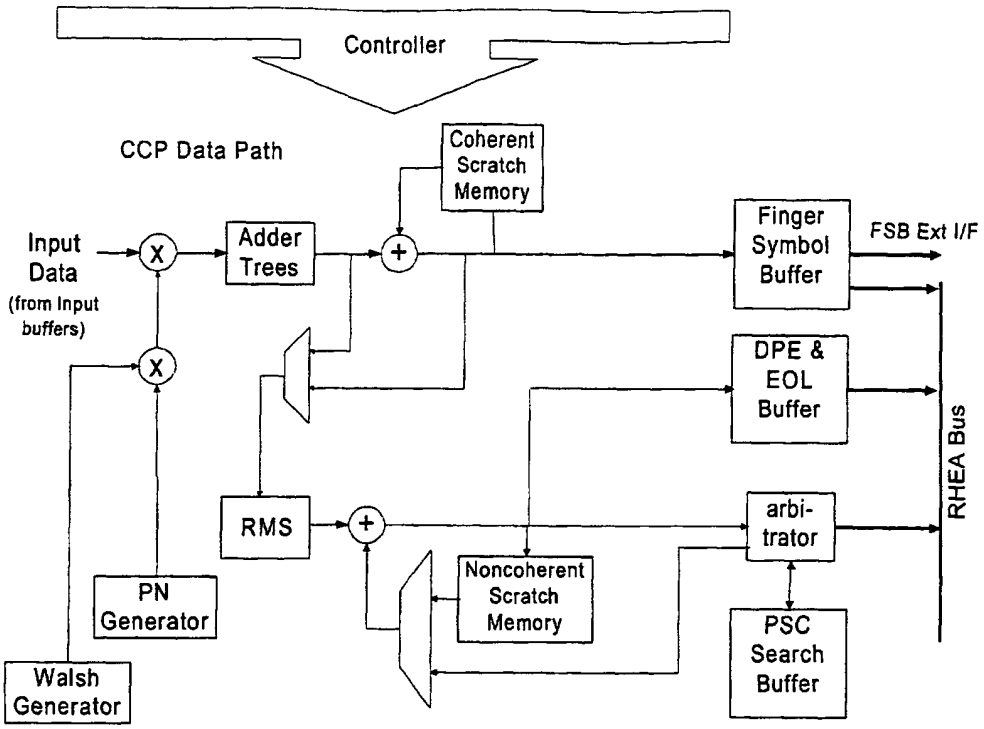


Figure 3. CCP Data Path

2.2.1 Datapath Precision

The CCP accumulates bits and discards bits at different stages of the datapath. The input from the A/D is 6 bits, and after the adder trees, there are 17 bits. At this point, some bits are discarded. Before writing symbols to the Finger Symbol Buffer, 9 MSB's are discarded, with saturation, for SF = 4; or 1 MSB is discarded, with saturation, for other SF's. For the symbols passed into the remainder of the datapath (EOL, DPE, LCI), 4 MSB's and 2 LSB's are discarded, with saturation.

After Coherent Accumulation, there are 22 bits. Of these 22 bits, 18 bits are kept starting from $(13 + \max(5, \log2(N_s)))^{th}$ LSB. Where N_s is the number of symbols of coherent accumulation.

After Non-Coherent Accumulation, there are 32 bits. Of these 32 bits, 24 bits are kept starting from $(13 + \max(5, \log2(N_{NS})))^{th}$ LSB. Where N_{NS} is the number of non-coherent accumulations.

2.3 PSC Search Buffer

The PSC Search Buffer serves two purposes. First, it stores running energy values while the PSC Search task is active. In this regard, it is used as accumulator memory by the CCP. Second, when the PSC Search task is finished, it stores the final energy values, which can be read by the host processor. One energy value per ½ -chip offset is returned. There are 5120 energy values.

While the PSC Search task is active, the PSC Search Buffer is accessible only by the CCP's datapath. When the PSC Search task is inactive, the PSC Search Buffer is accessible only via the RHEA bus. An arbitrator handles access rights, as shown in Figure 3. An interrupt may be generated when a PSC Search task finishes.

2.4 DPE and LCI Buffers

The DPE and LCI Buffers store DPE and LCI search results respectively. They are directly readable via the RHEA Bus at all times.

The DPE and LCI Buffers are single-buffered, and new results over-write old ones. When new results are ready, they may be read on the RHEA Bus directly by the host processor or by the RHEA DMA controller. Task-based interrupts can be generated when new results are ready. For example, when a DPE task finishes, an interrupt may be generated.

2.5 EOL Buffer

The EOL Buffer stores finger EOL measurement results. It is directly readable via the RHEA Bus at all times.

The EOL Buffer is single-buffered, and new results over-write old ones. When new results are ready, they may be read on the RHEA Bus directly by the host processor or by the RHEA DMA controller. The Finger task can issue various slot-based interrupt events that can be used to signal the availability of new EOL data.

2.6 Finger Symbol Buffer

The Finger Symbol Buffer stores complex I and Q "symbols" that result from Finger tasks. All symbols – Pilot, TPC, data, etc. – are stored here after they are received and processed by the CCP Datapath.

The Finger Symbol Buffer is implemented as a multi-slot circular buffer for each Walsh channel.

The Finger Symbol Buffer serves as intermediate storage for downstream symbol-rate processing. Its size is a compromise between area and the rate at which data must be moved to where downstream processing takes place. It is accessible on the RHEA bus. It is also accessible on the FSB External Bus, as shown in Figure 2. The FSB External Bus may be used when downstream processing and/or storage take place outside of the host processor (DSP system).

The Finger Symbol Buffer is described in detail in Section 7.

2.7 PN & Walsh Code Generators

A CCP task specifies the PN code (Gold code) and the Walsh code to be generated as well as the code offset. The PN/Walsh code generators then generate a block of the specified PN/Walsh codes starting from the specified code offset.

Gold code generation is centralized and can be produced for any correlation cycle. No LFSR state nor "mask" need to be specified, only the code number and offset from GCC. Both "block" and "serial" Gold code generation methods are employed to minimize power dissipation.

The l6x16 WCDMA PSC and SSC structures are programmable parameters. This specification is used for the PSC and SSC search operations.

2.8 Controller

The control part of the CCP is responsible for actually implementing each of the CCP tasks, and generating appropriate control signals for the datapath. Different flavors of correlations can be done by the datapath by varying the control sequence. When no tasks are running, downstream control and datapath pipeline stages are gated off to conserve power.

2.9 Global Chip Counter (GCC)

The local timing reference of the CCP is maintained by Global Chip Counter, or GCC, which counts the incoming chip samples as they are written into the Input Buffers. This counter counts modulo the length of the WCDMA long code (38400). All timing in the CCP is relative to the GCC, including offsets used in RAKE receiver operations.

2.10 Configuration Tables

The CCP uses a number of configuration tables to specify how it executes each of its tasks. Some tables are used globally, while others are associated with certain tasks. For

example, one configuration table contains the position and size of the Pilot symbols for each spreading factor. Another contains the Walsh codes associated with a particular Finger task. Configurations come directly from the host processor. All configuration tables are described in detail in Section 5.

2.11 Interrupt Generator

There are three types of interrupts in the CCP. They are task-based interrupts, system interrupts, and error interrupts.

Each CCP task can generate at least one interrupt. For example, when a DPE task finishes, it may generate an interrupt. Each Finger task can generate a number of interrupts, for example, to indicate the end of a radio slot or the reception of the TPC symbol. Task-based interrupts are mainly used by the host processor for data retrieval but may be for other SW/HW synchronization purposes.

Task-based interrupts place status information in one of four interrupt FIFOs. Each interrupt FIFO is tied to one of the four interrupt lines coming from the CCP.

System interrupts indicate global CCP events: The Task-Update interrupt signals the host processor that task updates are completed.

An error interrupt is generated whenever an error condition is detected.

CCP interrupts are described in detail in Section 8.

2.12 Task Buffer

The Task Buffer contains a list of tasks that the CCP executes. The Task Buffer is read directly by the CCP in order to determine the CCP's current tasks. The Task Buffer is a ping/pong buffer with an individual control for the ping/pong status of each entry in the Task Buffer. The swapping from ping to pong or vice-versa occurs on a Task-Update boundary. The Task-Update interrupt tells the host processor when the transfer completes, and that the updated status bits are available for each task. This mechanism allows a synchronization between the host processor and the CCP which prevents incomplete tasks being read by the CCP.

The CCP supports up to MAX_TASKS tasks in the Task Buffer.

3. Operations Overview

This section describes CCP operations: the concept of cycles and their management, programming the CCP, management of tasks, output memory, and interrupts.

3.1 Tasks

The CCP executes host-specified tasks. In a CCP iteration, which lasts 16 chips, all running tasks process 16 chips of data. Then in the next CCP iteration, the same tasks continue the processing – of the next 16 chips of data. Hence, a task typically runs for many CCP iterations. The CCP iteration boundaries occur when the Global Chip Count (GCC) is at a 16-chip boundary, that is, when GCC modulo 16 is 0. See Figure 4.

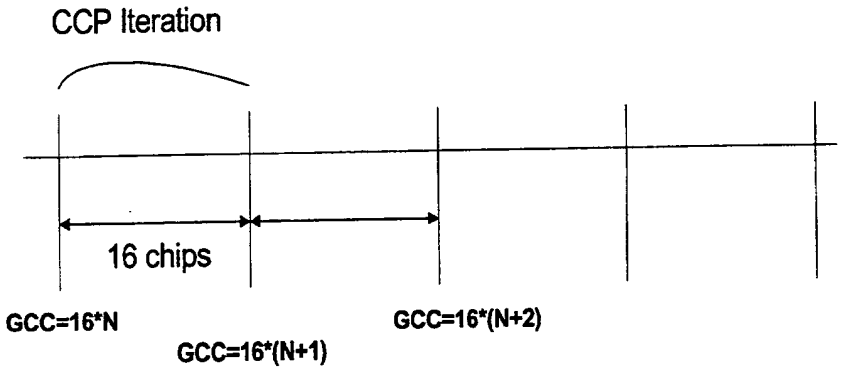


Figure 4: CCP Iteration

There are MAX_CYCLES CCP cycles available in each CCP iteration to execute tasks. These CCP cycles can be used for finger tasks, search tasks, etc. If there are extra cycles remaining, the CCP will automatically enter a power saving state.

In each CCP iteration, most tasks consume more than one CCP cycle. The number of cycles required by each task is described along with each task in Section 6.3.

New tasks are swapped-in and/or current tasks are modified at particular CCP iteration boundaries called Task-Update boundaries. Immediately following the Task-Update boundary, the CCP swaps the tasks, as requested, into the Task Buffer, then starts executing tasks from the first address of the Task Buffer. Each Task-Update boundary is associated with the Task-Update Interrupt event. Details are presented in Section 6.1.

Some tasks (e.g. LCI Search) require considerably more than one iteration pass to complete, and stop when they are completed. Others (e.g. Finger) run continuously until they are disabled by the host processor.

3.1.1 Cycle Management

It is the responsibility of the software running on the host processor to manage the available MAX_CYCLES CCP cycles. If the MAX_CYCLES cycles are expended and the CCP has not executed all enabled tasks, an Error interrupt event is issued.

In order to assist in cycle management, and for debugging purpuses, the Cycle_Count register is updated every CCP iteration by the CCP to indicate how many cycles were expended in the last CCP iteration. This register will change infrequently as tasks are added or completed. To ensure that an accurate value is read from this register, it should be read directly following the Task_Update boundary.

3.1.2 Task Management

Up to MAX_TASKS tasks can be stored in the Task Buffer. They are acted on in numerical order by address. The order of placement of tasks in the Task Buffer does not affect the results of a task in any way, with the exception that the PSC Search task (if present) must be the first task to execute. The CCP processes only the tasks that are "enabled," as discussed in Section 6.2.

After the CCP finishes executing all of the enabled tasks, its data path will be shut down for the remaining cycles in the iteration, and reads and writes to data path memory will cease. Some of the control logic will remain operational. This will result in significant power savings within the CCP.

3.1.3 Adding New Tasks

New tasks may be added at any time by writing them to the Task Buffer and then requesting an immediate load for that task.

The configuration memories and registers must be initialized before a task which references them is programmed. A task may actively run in the first CCP iteration that follows the load of a task into the hardware side of the Task Buffer. Therefore, configuration parameters for a task must be programmed before, or *on* the Task_Update boundary that loads the task.

3.1.4 Starting and Stopping Tasks

Having the task specification in the Task Buffer is not enough for the task to execute, the task must also be enabled by requesting either a synchronous start or an immediate start. An immediate start takes effect at the Task-Update boundary on which the request is received, and a synchronous start takes effect at the slot number specified for that task (which is with reference to the task's own slot timing). Each task can be enabled at any Task-Update boundary. There is no limit on the number of tasks in the Task Buffer that may be enabled, except in the total cycles consumed by the ones enabled.

The same concept applies to stopping tasks. All tasks can also be requested to stop immediately or synchronously at any Task-Update boundary.

3.1.5 Modifying Running Tasks

The mechanism to modify tasks is the same as loading tasks. To change a task's parameters, the whole task specification must be written into the Task Buffer and then an immediate load requested; no change to the enable status is made. In addition, a task may be synchronously (to its own slot timing) be reloaded.

Changing task parameters must be done with great care because the new parameters may conflict with the previous ones, thus producing erroneous results. Which parameters may be safely changed for each task is discussed in Section 6.3.

3.1.6 Task Run/Stop Status

The CCP keeps a status register indicating the run/stop state of each task. A task is in one of four states: stopped, waiting to run, running, or waiting to stop. An "enabled" task is one that is running, waiting to run, or waiting to stop.

Once the task begins executing, it will execute forever (until stopped by SW) or until a well-defined end time. The stopping of a task can be done by software manually, or will be done by the CCP automatically in the case of a task with a well-defined end-time. Automatic de-activation (stopping) times are described along with each task's description in Section 6.3. A "disabled" task is one that is in the stopped state.

3.2 Configuration Parameters

Configuration parameter are information used by running tasks. Some parameters are expected to be changed while a task is running, but most are not. There are provisions to allow changes to those parameters which may need to change while the task is running so that the changes have predictable results. The parameters which have such provisions are

the Finger Interrupt Table, the Finger Symbol Buffer Configuration Table, and the Walsh Table.

Entries in the Finger Interrupt Table are transferred from the software side of the double buffer, to the hardware shadow of the double buffer at each Task_Update boundary.

Entries in the Walsh Table and Finger Symbol Buffer (FSB) Configuration Table, may be written to at any time. Pointers to the tables are used by the tasks, and it is these pointers which can be changed while the task runs with predictable behavior. It is the responsibility of the software to avoid writing to any entry which is currently in use by the CCP. Should this happen, the results are undefined. Refer to Section 6.1.

3.3 Initializing the CCP

After a power-on reset, all the CCP registers are in a known state, the CCP memories have contents which are undefined, and the CCP itself is not running any tasks.

In order to configure the CCP to run its first tasks, the following sequence of steps is recommended, once the clock to the CCP has been started:

1. Configuration memories and registers should be programmed (e.g. Walsh Table, Finger Symbol Buffer Configuration Table, Finger Interrupt Table, Task_Update_Cycle register, Pilot Bits Table, Interrupt etc.)
2. Tasks should be written to the Task Buffer.
3. Task Requests (Start, Load) should be written to their respective registers.
4. Then the CCP should be started with the Start/Continue command.
5. At the first Task_Update boundary, all specified Task Requests (Start, Load) will occur and the tasks will begin running.

The contents of all memories is undefined and must be initialized before being referenced. The entire Task Buffer need not be initialized before use, just the locations which are to be used. All tasks are disabled at reset.

Task Update interrupts begin when the CCP is enabled to run via a start command or by a sleep timer.

3.4 Finger Symbol Buffer Management

The Finger Symbol Buffer is set up as multi-slot circular buffer areas for each despreader (sub-task) which uses them, as specified in the FSB Configuration Table. It is the responsibility of the software running on the host processor to manage these areas, and in keeping the areas from overlapping.

When the CCP is writing to a particular address in this memory, the reading of that address would result in an undefined data word being read. For this reason, the SW must react to interrupts in a timely way so as to always read in "safe areas". Further details are specified in Section 7.

3.5 Interrupts

Many events in the CCP may cause interrupts on one of the external interrupt lines. There are system interrupt events, error interrupt events, and task-based interrupt events. Each interrupt event can be independently. A FIFO queuing system is implemented for the task-based interrupts, so that several pending interrupts can be held until the processor services them. This is described further in Section 8.

4. External Interface

This section describes the external interfaces to the CCP which includes a RHEA bus interface, the Finger Symbol Buffer (FSB) external bus, I/Q data inputs, and other control connections.

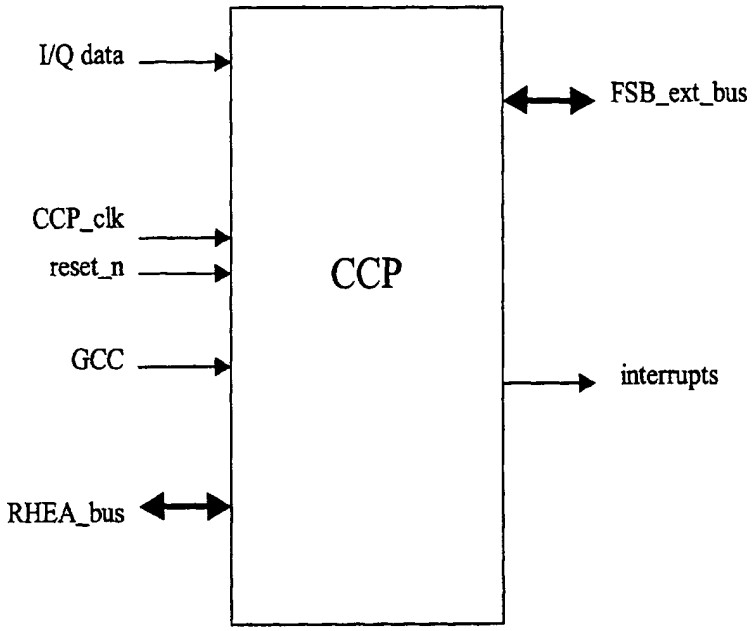


Figure 5. CCP External Interface

4.1 Interface Signal Description

DSP RHEA BUS		
XAD(9:0)	IN	Address bus
XCS(4:0)	IN	Chip Select

XDO(31:0)	IN	Data from RHEA bridge to peripherals
XDI(31:0)	OUT	Data from peripherals RHEA bridge
XRnW	IN	Read not Write
NXSTROBE(1:0)	IN	Strobe lines
NXREADY	OUT	Peripheral Ready to accept or to send data
NXSUSPEND	IN	Indicates that DSP has suspended execution for an emulation breakpoint.
XPERHMAS	OUT	Accessed Peripheral register size 0 => 8 bits, 1=> 16 bits
XMAS	IN	Memory access length 0 => 8 bits, 1=> 16 bits
XIDLE	IN	Idle peripheral
nXIRQ(11:0)	OUT	LEAD Interrupt requests.

FSB External Bus

FSB_AD(13:0)	IN	FSB Address inputs
FSB_DATA(31:0)	OUT	FSB Data word
FSB_RD_REQ_N	IN	FSB Read request, active low
FSB_READY_N	OUT	FSB Data ready, active low

Control Signals

RESET_N	IN	Reset signal

Interface to System Controller		
GCC[18:0]	IN	Global Chip Count, System Time Base
CCP_enable	IN	CCP enable
Bdry_16_chip	IN	16 chip boundary
CCP_clk	IN	clock
CCP_Int_n[3:0]	OUT	Interrupts, active low
System_Int_n	OUT	System interrupt, active low
Error_Int_n	OUT	Error interrupt, active low

Interface to IO Select		
Ot_sample[1:0]	I	On-time sample number
Lt_sample[1:0]	I	Late-time sample number
Ot_frame[3:0]	I	On-time frame location, 1 of 16 chips
Lt_frame[3:0]	I	On-time frame location, 1 of 16 chips
Ot_data[15:0]	I	On-time data, 16 samples
Lt_data[15:0]	I	Late-time data, 16 samples

5. Software Interface

This section covers the following topics:

1. Task Buffer
2. Synchronously Updated Configuration Parameters
3. Asynchronously Updated Configuration Parameters
4. Command Set
5. Global Status
6. Interrupt Status
7. FSB Status
8. Output Data

Note 1: For compatibility with future versions of the CCP device, all bit fields marked "unused" or "value immaterial" should be filled with zeros.

Note 2: All data values are oriented with the LSB at bit 0, and the MSB at the highest bit number.

Note 3: When a 32 bit field is read over a 16 bit data bus, the address of for bits 31 – 16 is even and preceeds the address for bits 15 – 0.

5.1 Task Buffer

The Task Buffer is a collection of MAX_TASKS ping/pong buffers each of which may contain a CCP task of 90 bits (5*16). The status of each of MAX_TASKS tasks as to whether the ping or pong is available to the CCP for exectution, and the other available for SW loading/modification, can be controlled on an individual task basis by SW. Task formats are found in Section 6.3. Entries modifiable by the SW may be read by host processor.

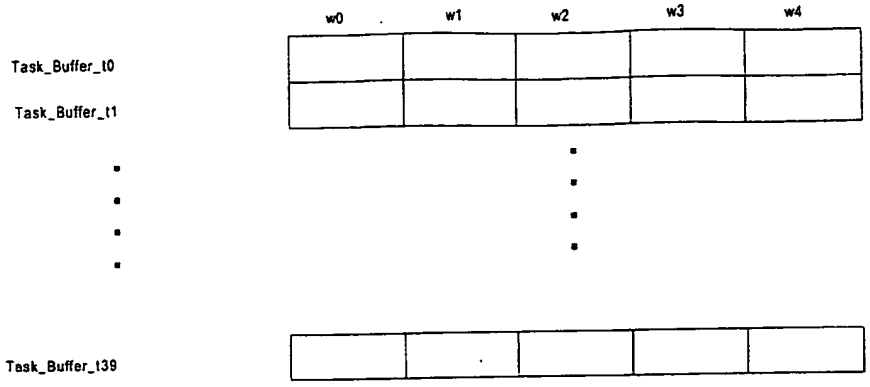


Figure 6: Task Buffer

	15 – 0 (R)
Task_Buffer_t0_w0	First Word of Task Buffer Entry (MSW)
Task_Buffer_t0_w1	Second Word
Task_Buffer_t0_w2	Third Word
Task_Buffer_t0_w3	Fourth Word
Task_Buffer_t0_w4	Last Word of Task Buffer Entry (LSW)

Figure 7: Task Buffer Entries

5.2 Synchronously Updated Configuration Parameters

These configuration parameters are double buffered and are updated at the Task-Update boundary: the software-modified register is copied to the hardware-readable shadow register at this time.

5.2.1 Task Request Bits

Tasks Requests to Start/Stop, Load/Reload, or Adjust Timing may be made at the next Task-Update boundary (following a write access) by setting the appropriate Task Request Bits. A "1" indicates a new request, '0' indicates no new request (requests may still be pending). The requesting will continue at succeeding Task-Update boundaries until the request is cleared by SW, via a "0", therefore, for proper operation, a "0" should be written at the Task-Update boundary immediately following a "1" being written. Refer to Figure 8. An operational description of Task Request is found in Section 6.2.1.

	15 – 0 (R/W)
Task_Req_w0	Request for tasks 15 – 0
Task_ Req_w1	Request for tasks 31– 16
Task_ Req_w2	Request for tasks 47– 32
Task_ Req_w3	Request for tasks 63– 48

Figure 8: Task Request Bits

The type of request is indicated by writing the appropriate information to the task's Task Request ID field.

5.2.2 Task Request IDs

Each task has a Task Request ID register which identifies the type of Request being made when a "1" is written to the corresponding Task Request Bit. The format of the Task Request ID is:

	9	8 - 7	6	5	4	3 - 0 (R/W)
Task Request ID	Start/Stop Enable	Start/Stop ID	Load/ Reload Enable	Load/ Reload ID	Timing Adjust Enable	Slot #

Figure 9: Task Request ID Register Format

Start/Stop Enable: a "1" indicates that the request includes the start/stop action identified in the Start/Stop ID bits, a "0" indicates no start/stop action is requested.

Start/Stop ID: These bits are referenced by the hardware only if the Start/Stop Enable bit is set ("1"). They have the following meaning:

- "00" indicates Immediate Start
- "01" indicates Synchronous Start
- "10" indicates Immediate Stop
- "11" indicates Synchronous Stop

Load/Reload Enable: a "1" indicates that the request includes a load/reload action as identified in the Load/Reload ID bit, a "0" indicates no load/reload action is requested.

Load/Reload ID: a "0" indicates an Immediate Load is requested, a "1" indicates a Synchronous Reload is requested. This bit is referenced by the hardware only if the Load/Reload Enable bit is set ("1"). A synchronous Reload implies the instruction is

"enabled", if it is not, the Reload will remain pending until an Immediate Load is requested.

Timing Ajust Enable (for Finger Tasks only): a "1" is a request to adjust timing, and "0" is not a request to adjust timing. The timing change information is contained within the Finger Task description.

Slot #: This field identifies the slot number for a synchronous action (synchronous start, synchronous stop, or synchronous reload). When a synchronous start is requested, this field defines the first slot to begin processing. When a synchronous stop is requested, this field defines the first slot which is not processed, in other words, the last slot number processed is the one which precedes this slot number (modulo 15). When a synchronous reload is requested, this field defines the first slot after the reload, in other words, the last slot number processed before the reload is the one which precedes this slot number (modulo 15).The fifteen slots per frame are numbered 0 through 14. The value of 15 in this field indicates the beginning of the next slot. Any request for a given which references this value must complete before another request references this value. Otherwise, they will both be referencing the last value written.

5.2.3 Walsh Table

The Walsh Table holds Walsh sub-tasks. A set of Walsh sub-tasks is used by a Finger or DPE task to specify how Walsh de-spreading should take place, on one or more (in case of Finger) Walsh channels. Each Finger or DPE task has a Walsh Pointer field to specify the desired set of Walsh sub-tasks. In addition, Finger tasks may specify the processing of an EOL sub-task which will support the DLL function by computing on-time, early and late energies on a particular Walsh channel.

The Walsh Table is made up of TOTAL_WALSH different sections or sets, and each is configured with a fixed number of Walsh sub-tasks. The first NUM_WALSH8 sets of Walsh sub-tasks, addressed by the first Walsh pointers 0 through (NUM_WALSH8 – 1), may each specify up to 8 Walsh sub-tasks, and the last NUM_WALSH4 sets, addressed by Walsh pointers NUM_WALSH8 to (NUM_WALSH8 + NUM_WALSH4 – 1), may each specify up to 4 Walsh sub-tasks. The TOTAL_WALSH sets of Walsh sub-tasks may specify a maximum of (NUM_WALSH8 * 8 + NUM_WALSH4 * 4) Walsh sub-tasks.

The CCP supports up to MAX_FINGERS Finger tasks using MAX_FINGERS unique Finger ID's. Each Finger task requires a set of Walsh sub-tasks with 1 to 8 valid entries. Finger tasks with the first NUM_LARGE_FINGERS Finger ID's constrained to point to a set of sub-tasks of size 8, and remaining Finger ID's constrained to point to one of set of sub-tasks of size 4. More than one Finger task can point to the same Walsh sub-task set.

A DPE task requires a single Walsh sub-task which can be located in any set of Walsh sub-tasks as the first sub-task in that set, (Any Walsh pointers may be used).

An example of 64 sets of Walsh sub-tasks, with NUM_WALSH8 = 32 and NUM_WALSH4 = 32, is shown in Figure 10.

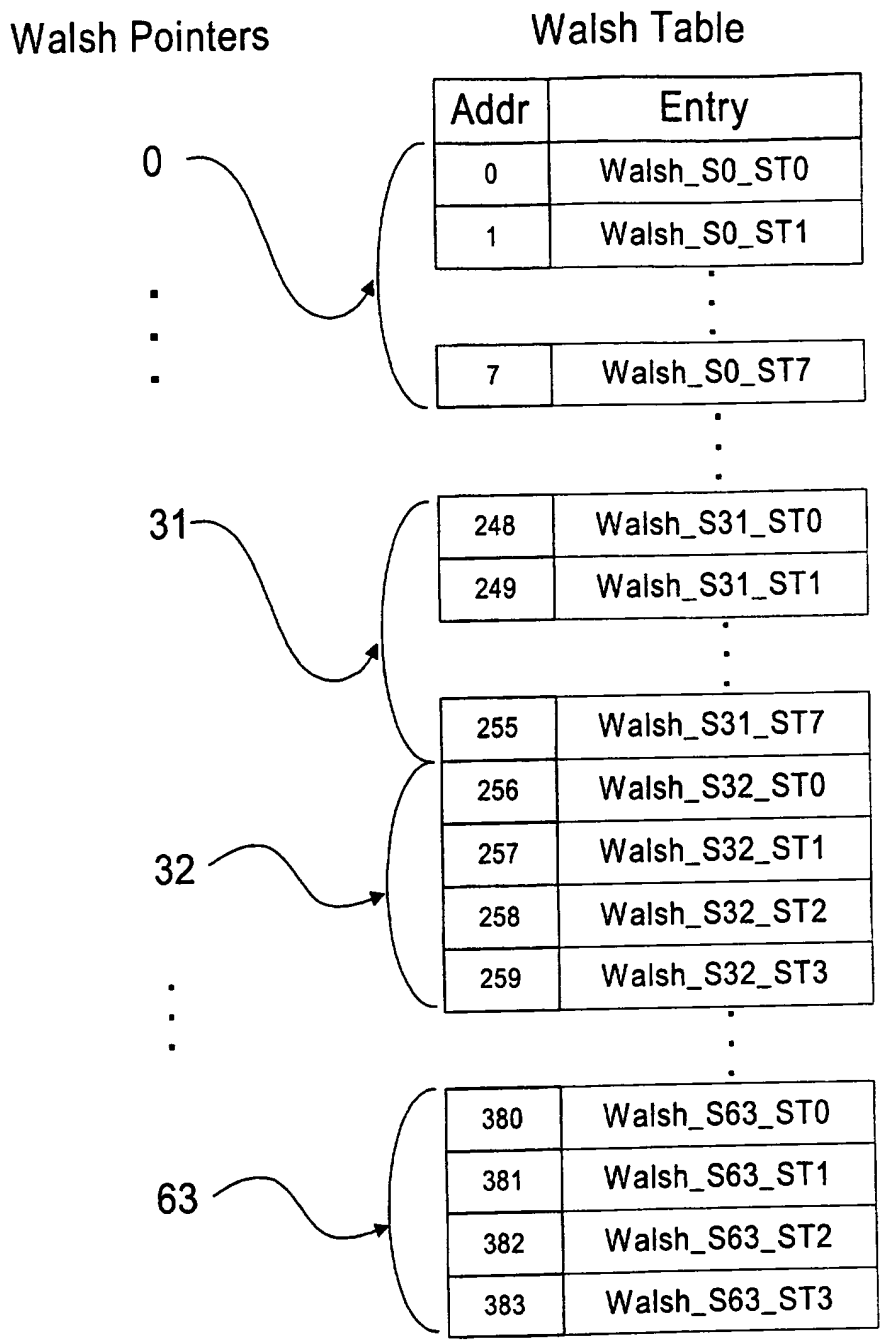


Figure 10: Walsh Table

If fewer than the maximum number of entries (8 or 4) are needed, the sub-tasks must appear at the beginning of the area and be consecutive. If there is an EOL Walsh sub-task, it must be at the first location.

Each sub-task has a Walsh ID which must be unique in the set. In sets of four Walsh sub-tasks, only ID's 0 through 3 are valid, in sets of eight Walsh sub-tasks, ID's 0 through 7 are valid. The Walsh ID is chosen by the host processor and allows for flexible reconfiguration of Walsh sub-tasks while a Finger task is running.

The Walsh Table sub-tasks are entered using the Temporary Buffer. Its contents are readable by the host processor. A sample Walsh sub-task is shown in Figure 11 that belongs to the fifth Walsh set (S5) and is the third Walsh sub-task (ST3).

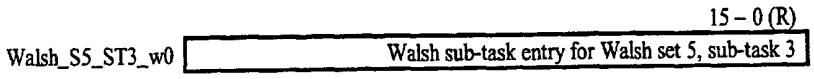


Figure 11: Walsh Table Entry

5.2.4 Finger Symbol Buffer Configuration Table

The FSB Configuration Table is a ping/pong buffer with two entries for each Finger ID/ Walsh ID (despreader) combination, to specify where in the Finger Symbol Buffer a finger and each of its Walsh channels (as specified by the Walsh ID) will output its de-spread symbols. The first NUM_LARGE_FINGERS fingers may specify up to eight Walsh ID's, and each of the remaining fingers may specify up to four Walsh ID's. Hence, there are 8*NUM_LARGE_FINGERS + 4*(MAX_FINGERS - NUM_LARGE_FINGERS) entries. Figure 12 shows an example with MAX_FINGERS = 64, NUM_LARGE_FINGERS = 32. Each entry consists of the start address for the first slot of data in the four-slot circular buffer and the address offset from one slot to the next slot. Further details are found in Section 7.2.

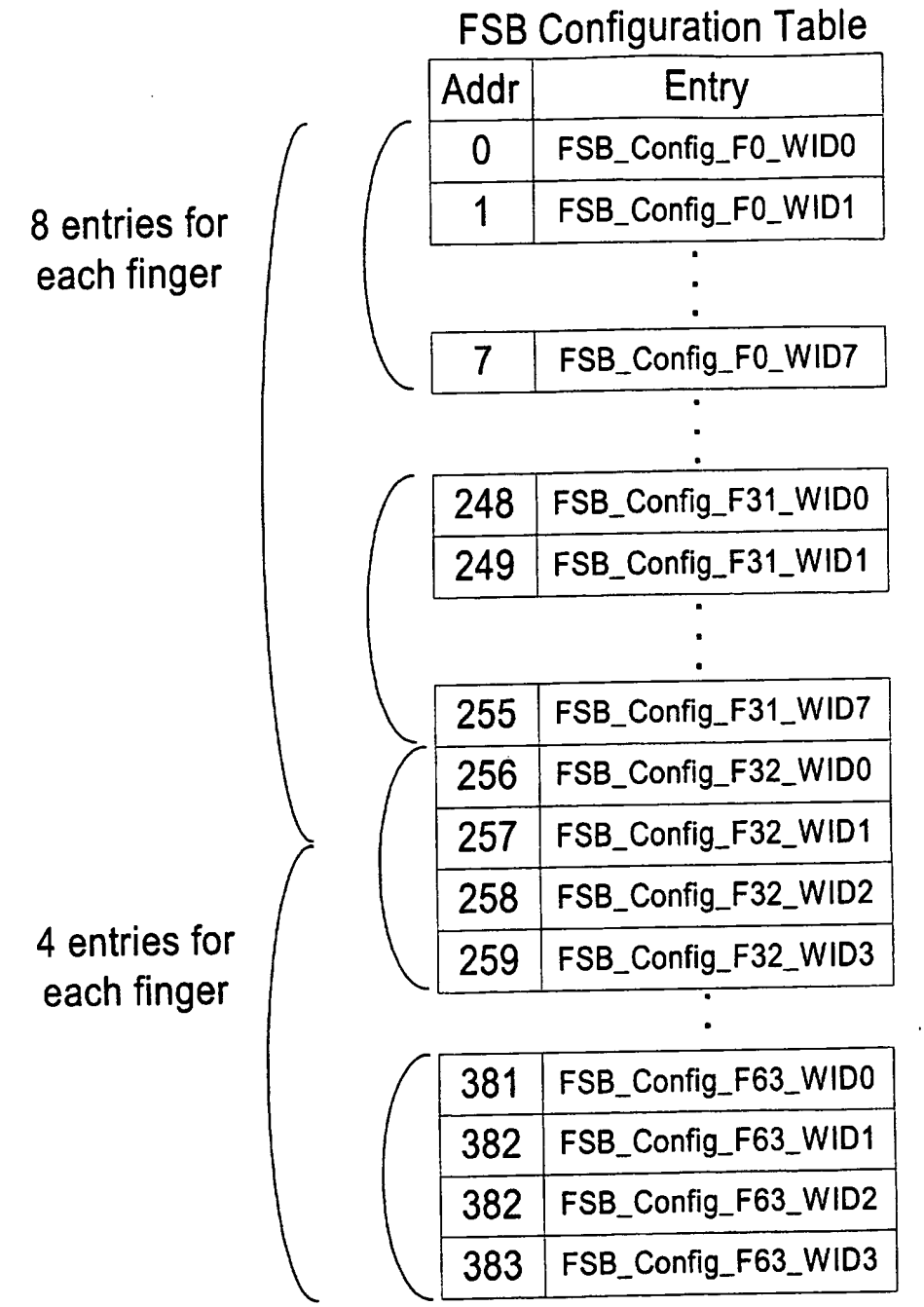


Figure 12: FSB Buffer Configuration Table

The FSB Configuration Table is written to directly, with the ping and pong sides written independently. Each entry is 32 bits wide. The entries are readable as shown in Figure 13.

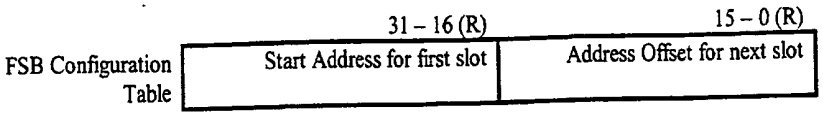


Figure 13: FSB Buffer Configuration Table Entry

5.2.5 Finger Interrupt Table

The Finger Interrupt Table associates a Finger ID with a set of interrupt events. Each Finger ID has a location in the table that can enable the following interrupts: Slot Interrupt, TPC Interrupt, and Pilot Interrupt. The size of the table is MAX_FINGERS x 9 bits.

The Finger Interrupt Table is double buffered. The contents of the table, which SW writes to, are copied to the CCP hardware view at the Task Update Boundary. The table, as viewed by the CCP hardware, is readable by the host processor. The format of the table is as shown in Figure 14.

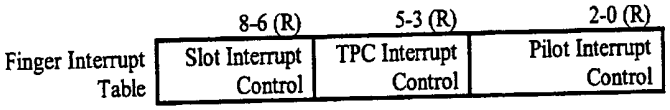


Figure 14: Finger Interrupt Table Format

Each of the 3-bit fields in a Finger Interrupt Table entry (Slot, TPC, and Pilot Interrupt Control) shown in Figure 14 have a common format, as shown in Figure 15, where the field for controlling the Pilot Interrupt is shown. The MSB controls enable status, and the 2 LSB's specify the interrupt FIFO.

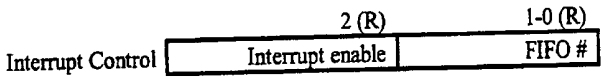


Figure 15. Finger Interrupt Control Format

5.2.6 Task_Update_Cycle Register

The Update_Cycle register identifies the how often there is a Task_Update event. It is implemented with a shadow register which is updated on the Task_Update boundary. At reset, the value of Update_Cycle is 4 decimal.

	15 – 0 (R/W)
Update_Cycle	Task-Update Cycle

Figure 16: Task_Update_Cycle Register

5.3 Asynchronously Updated Configuration Parameters

Asynchronously updated configuration parameters are typically ones that are not changed often, if at all, once the CCP begins its operations. Changing these parameters when not in use is not a problem (eg. Changing the DPE/LCI Data Truncation register when no DPE or LCI tasks are running).

5.3.1 Pilot-TPC Position Table

The Pilot-TPC Position Table specifies the position of Pilot and TPC symbols. There are eight (8) entries, each corresponding to a spreading factor from 4 to 512. Each entry specifies two separate locations for Pilot symbols and one location for the TPC symbol, as shown in Figure 17.

The TPC position field specifies the symbol number of the TPC symbol in a radio time slot.

Each Pilot position field specifies a contiguous region of symbols within the radio time slot. The region is specified using a starting position and an ending position, as shown in Figure 17.

	9 – 0 (R/W)
Pilot_TPC_Pos_SF0_w0	Starting symbol # for Pilot region 0
Pilot_TPC_Pos_SF0_w1	Ending symbol # for Pilot region 0
Pilot_TPC_Pos_SF0_w2	Starting symbol # for Pilot region 1
Pilot_TPC_Pos_SF0_w3	Ending symbol # for Pilot region 1
Pilot_TPC_Pos_SF0_w4	TPC Symbol #

Figure 17: Pilot-TPC Position Table Entry Format

In actuality, the Pilot regions defined are not only used to specify the location of Pilot symbols; their usage depends on the task and sub-task that use them. This is why there are two defined Pilot regions for each spreading factor.

The Finger task selects one of these Pilot regions to specify when the Finger Pilot Interrupt is activated, which is at the end of the last symbol in this region. For this particular purpose, it is only the end of the region that is important.

An EOL Walsh sub-task or a DPE Walsh sub-task can select one of the Pilot regions in the Pilot-TPC Position Table to define which symbol energies to accumulate. Here, the defined region does not have to correspond exactly to the forward-link Pilot-symbol region, for example, when it is desirable to measure the energies of additional non-Pilot symbols. Alternately, these sub-tasks may measure the energies of symbols outside of the selected Pilot-symbol region.

5.3.2 Pilot Bits Table

The Pilot Bits Table has an entry for the modulation of the pilot bits (up to 16 bits per slot are supported) for each radio slot (up to 16 slots) for four different options. The common pilot bits (on the Common Pilot Channel CPICH in IMT2000-DS) need not be entered in the Pilot Bits Table, they are available as option 0, and their values are hardwired. The pilot bits for the diversity antenna are stored along with the pilot bits for the first antenna. The size of the table is (3 x 16) entries by 32 bits. The Pilot Bits Table is used by Finger-EOL, and DPE tasks.

	31 – 16	15 – 0 (R/W)
Pilot_Bits_1_Slot0	Pilot Bits for option 1, slot 0	Diversity Pilot Bits for option 1, slot 0
Pilot_Bits_1_Slot1	Pilot Bits for option 1, slot 1	Diversity Pilot Bits for option 1, slot 1
.....	.....	.....
Pilot_Bits_1_Slot15	Pilot Bits for option 1, slot 15	Diversity Pilot Bits for option 1, slot 15

Figure 18: Pilot Bits Table Entry Format

Each Pilot bits entry, which encodes one slot, contains 16 bits for each antenna. In WCDMA, each pair of bits is a complex pilot symbol. The first pair of bits in the slot resides in bits 31 and 30, with bit 31 being the in-phase bit and bit 30 being the quadrature-phase bit. The next pair of bits is in bits 30 and 29, and so on. The number of Pilot bits in an actual radio slot may be less than 16, in which case the first n bits are used, where n is the total number of bits required.

Each task which requires the use of the Pilot Bits Table will specify the Pilot Bits option number, along with a Pilot region, which together indicate the number of pilots and their values.

5.3.3 External Interrupt Enable Register

The External Interrupt Control register controls the enabling of FIFO, error and system interrupts to the external interrupt lines.

	15 - 6 (R/W)	5 (R/W)	4 (R/W)	3 - 0 (R/W)
External_Interrupt_Enable	unused	System Interrupt Enable	Erorr Interrupt Enable	FIFO Interrupt Enables

Figure 19: External Interrupt Enable Registers Format

5.3.4 PSC Register

The First Search Code has a hierarchical structure defined by the Kronecker product of two 16-bit sequences, PSC = PSC0 X PSC1, where "X" denotes the Kronecker product.

	15 – 0 (R/W)
PSC0	PSC0
PSC1	PSC1

Figure 20: PSC Register Format

5.3.5 SSC Register

The Secondary Search Code has a hierarchical structure defined by the Kronecker product of two 16-bit sequences, SSC = SSC0 X SSC1 followed by the modulo-2 addition with a Hadamard sequence, where "X" denotes the Kronecker product. The SSC search task uses SSC0, the post-processing with SSC1 and the Hadamard sequence is left as a task for software or special purpose hardware.

	15 – 0 (R/W)
SSC0	SSC0

Figure 21: SSC Register Format

5.3.6 DPE and LCI Energy Accumulation Parameters

DPE and LCI tasks output results once, after measuring N_{NTS} * N_{TS} radio time slots of data. When Pilot symbols are accumulated coherently for longer than a symbol, N_{TS} controls the number of radio time slots of coherent accumulation. N_{TS} can have any value between 1 and 31 (unlike the restrictions for the EOL sub-task). When Pilot symbols are accumulated with a coherent length of one symbol, N_{TS} is ignored, and results are output after N_{NTS} time slots. The CCP datapath is designed for the accumulation of up to 8 frames, exceeding this range with any combination of these parameters may result in errors due to overflow.

There is a table for DPE parameters of size MAX_DPE x 13 bits, and one for LCI parameters of size MAX_LCI x 13 bits. The parameters for each DPE and LCI Search ID are stored in a separate entry with the following format:

Figure 22: DPE and LCI Energy Accumulation Parameters

5.3.7 Search Code Symbol Location Register

Specifies which symbol of a Perch channel contains the primary and secondary search codes (PSC and SSC). Possible values are from 0 to 9.

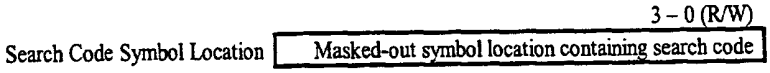


Figure 23: Search Code Symbol Location Register

5.4 Command Set

The following are memory-mapped commands: when SW writes to the address, the command is effective immediately, reading the address has undefined results.

5.4.1 Start / Continue Command

The Start/Continue command is used to start the CCP after power-on (see Section 3.3), and to continue the execution of the CCP after a Halt command. The CCP will always start on a 16-chip boundary of the Global Chip Counter (GCC). The step value defines the number of cycles to run before stopping, a value of "0" indicates to run continuously.

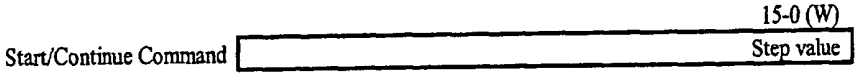


Figure 24: Start/Continue Command

5.4.2 Halt Command

The Halt command forces the CCP to halt on the next 16 chip boundary. Processing can be resumed with the Start/Continue command but there will be a break in the input stream which must be accounted for by SW. For proper operation, the procedure for programming the CCP after power-on should be followed (see Section 3.3).

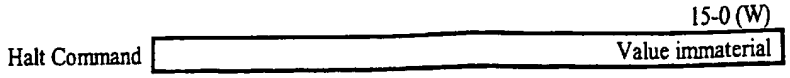


Figure 25: Halt Command

5.4.3 Software Reset Command

When the CCP is stopped or halted, this command will reset all internal registers and states to the power-on reset configuration. This command is not intended to be used while the CCP is executing.

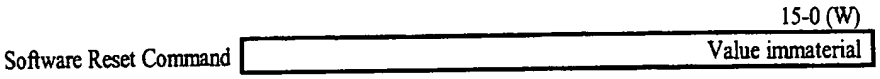


Figure 26: Software Reset Command

5.5 Global Status

A synchronizer circuit allows the processor to read accurate values from these registers.

5.5.1 CCP_Status Register

The CCP_Status Register contains

- PSC Buffer Status. 1 bit. "1" if PSC Buffer is accessible only by the CCP. "0" if PSC Buffer is accessible by host processor.
- CCP Run/Stop Status. 1 bit. "1" if CCP is running, "0" if stopped.

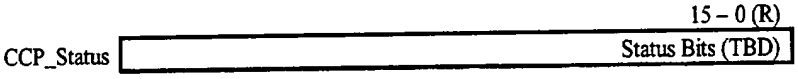


Figure 27: CCP_Status Register Format

5.5.2 Task Run/Stop Status Bits

The host processor can read the current Run/Stop status for all tasks in Task Buffer.

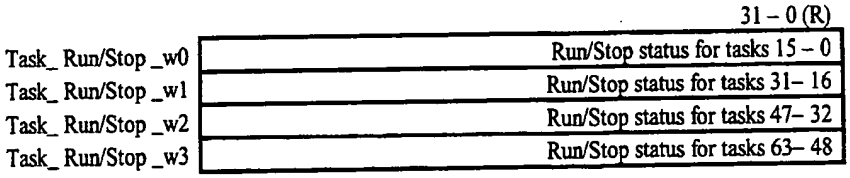


Figure 28: Task Run/Stop Status Bits

Each task has a two bit field which endodes the Run/Stop status in the following manner:

"00" indicates "Stopped"

"01" indicates "Waiting to Run"

"10" indicates "Running"

"11" indicates "Waiting to Stop"

5.5.3 Task Ping/Pong Status Bits

The host processor can read the current ping/pong status for all tasks in Task Buffer. "0" indicates that HW is reading ping and SW reading/modifying pong, "1" indicates the reverse.

Figure 29: Task Ping/Pong Status Bits

5.5.4 Task_Update_Timestamp Register

The Task_Update_Timestamp register captures the GCC value of the most recent Task_Update boundary.

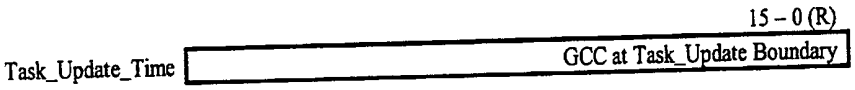


Figure 30: Task Update Time Register

5.5.5 Cycle_Count Register

The Cycle_Count register captures the number of cycles expended in the most recent CCP iteration; it is updated every CCP iteration.

	7 – 0 (R)
Cycle_Count	Number of Cycles Expended in Iteration

Figure 31: Cycle Count Register

5.5.6 GCC Count Register

The GCC Count register captures the current GCC value, updated once per chip. The 4 MSB's give the GCC local slot number and remaining LSB's give the rest of the count in chips.

	15 – 0 (R)
GCC Count	Current GCC value

Figure 32: Current GCC Count Value Register

5.6 Interrupt Status

A synchronizer circuit allows the processor to read accurate values from these registers.

5.6.1 Int_Error_Event_Status Register

Reading the Int_Error_Event_Status clears the Error interrupt event.

- Cycles Exceeded Error. 1 bit. "1" if the number of cycles attempted in an iteration was greater than the maximum (320), "0" otherwise.
- Interrupt FIFO Overflow Error. 4bits. One bit for each of the four interrupt FIFO's, "1" if the corresponding FIFO has overflowed, "0" otherwise.

	15 – 6 (R)	5 (R)	4 (R)	3 (R)	2 (R)	1 (R)	0 (R)
Int_Error_Event_Status Register	unused	FIFO3 Ovflw	FIFO2 Ovflw	FIFO1 Ovflw	FIFO0 Ovflw	Temp Buffer	Cycles Exceed

Figure 33: Int_Error_Event_Status Register

5.6.2 Int_System_Event_Status Register

Reading the Int_System_Event_Type clears the System interrupt event.

- Task_Update Event. 1 bit. "1" if the Task_Update event occurred, "0" otherwise.

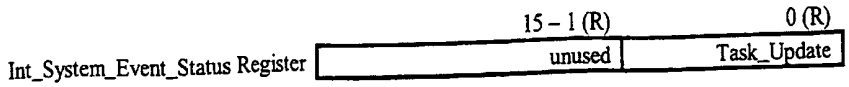


Figure 34: Int_System_Event_Status Register

5.6.3 FIFO Status

The FIFO Status register shows each interrupt FIFO's empty/non-empty status. When a particular FIFO is not empty, its FIFO Empty Status is "1". The status bit is cleared when all FIFO contents have been read out.

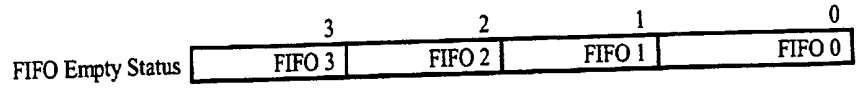


Figure 35: FIFO Empty Status

FIFO contents are 2 words each, and are read out at one memory location, one after the other. When the second word is read out, the FIFO hardware increments its internal read pointer.

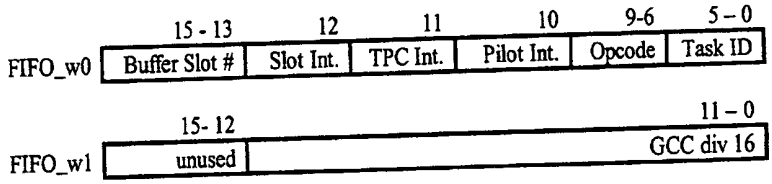


Figure 36. FIFO Content Format

The Finger task may issue multiple and simultaneous interrupt events for the slot, TPC, and Pilot, which are indicated by individual status bits in the first word. The Buffer slot number field indicates which slot in the multi-slot circular buffer that a Finger task or SSC Search task has placed its data; field is valid on when the Slot Interrupt bit field is set for a Finger task, and on all SSC Search task interrupts.

The second word of the FIFO records the GCC value when the event occurred.

5.7 Output Data

5.7.1 Finger Symbol Buffer (FSB)

The Finger Symbol Buffer holds 18K 32 bit words and is divided into multi-slot circular buffers for each Finger ID/ Walsh ID combination. The location of each circular buffer is defined in the FSB Configuration Table. Refer to Section 7. When a Finger Task has a spreading factor between 8 and 512 the following is the format of the data:

	31 – 16 (R)	15 - 0(R)
FSB Entry	Symbol (real part)	Symbol (imaginary part)

Figure 37: Finger Symbol Buffer (FSB) Format

When a Finger Task has a spreading factor of 4, the following is the format of the data:

	31 – 24 (R)	23 - 16(R)	15 - 8 (R)	7 - 0(R)
FSB Entry, SF=4 only	Even Symbol (real part)	Even Symbol (imaginary part)	Odd Symbol (real part)	Odd Symbol (imaginary part)

Figure 38: Finger Symbol Buffer (FSB) Format, SF=4 only

5.7.2 Finger Max Buffer

The Finger Max Buffer stores the largest energy value within a slot for a particular each Finger ID/ Walsh ID (despreader) combination in four slot circular buffer. Each energy value is 16 bits. There are 8*NUM_LARGE_FINGERS + 4*(MAX_FINGERS - NUM_LARGE_FINGERS) four slot circular buffers. They are arranged in the same order as the FSB Configuration buffer, and the four slot energies are grouped together. The buffer number (0-3) used for a particular slot is the same as the buffer number used by the FSB to store symbols, and the Finger Task interrupt information provides this buffer number in the FIFO entry.

	15 – 0 (R)
Finger Max Buffer Entry	Energy of largest symbol in slot

Figure 39: Finger Max Buffer Format

5.7.3 EOL Buffer

The EOL Buffer stores early, on-time, late energy measurements of Finger tasks. Results are dumped into the EOL Buffer once per frame -- at the end of each frame -- and must be

retrieved by the host processor before the next frame boundary. When new results are ready, they may be read on the RHEA Bus directly by the host processor or by the RHEA DMA controller.

The outputs are indexed by the Finger ID, as shown in Figure 40.

	31 - 24 (R)	23 – 0 (R)
EOL Buffer F0 dw0	unused	Finger 0: Early Energy
EOL Buffer F0 dw1	unused	Finger 0: On-time Energy
EOL Buffer F0 dw2	unused	Finger 0: Late Energy
.....	.....	
EOL Buffer F63 dw0	unused	Finger 63: Early Energy
EOL Buffer F63 dw1	unused	Finger 63: On-time Energy
EOL Buffer F63 dw2	unused	Finger 63: Late Energy

Figure 40: EOL Buffer Memory Map

5.7.4 DPE Buffer

Output results from the DPE task are placed in the DPE Buffer, which holds up to MAX_DPE*32*2 energy values. The DPE Buffer is partitioned into equal-sized blocks, each with 32 locations, a location holding an on-time energy and (optionally) a late-time energy for an offset within the search window.

A DPE task with a certain DPE Search ID outputs its results starting at the same block number in the DPE Buffer. For a DPE task, if the search window is less than or equal to 32 chips, then all of its results are placed in one block of the DPE/LCI Buffer. If the search window is more than 32, the results are placed in the next adjacent blocks of the DPE/LCI Buffer. Results are stored sequentially from the starting to the ending offset.

	23 – 0 (R)
DPE Buffer Block0 dw0	Energy value for Block 0, chip-offset 0
DPE Buffer Block0 dw1	Energy value for Block 0, chip-offset ½
.....	.....
DPE Buffer Block0 dw62	Energy value for Block 0, chip-offset 31
DPE Buffer Block0 dw63	Energy value for Block 0, chip-offset 31 ½
.....	.....
DPE Buffer Block15 dw0	Energy value for Block 15, chip-offset 0
DPE Buffer Block15 dw1	Energy value for Block 15, chip-offset ½
.....	.....
DPE Buffer Block15 dw62	Energy value for Block 15, chip-offset 31
DPE Buffer Block15 dw363	Energy value for Block 15, chip-offset 31 ½

Figure 41: DPE Buffer Memory Map

5.7.5 LCI Buffer

Output results from the LCI tasks are placed in the LCI Buffer, which holds MAX_LCI pairs of energy values. The LCI Buffer is partitioned into equal-sized blocks, each with 8 locations, a location holding an on-time energy and (optionally) a late-time energy for a long code.

An LCI task with a certain LCI Search ID outputs its results starting at the same block number in the LCI Buffer. The LCI task ouputs a pair of energies for each of up to 8 codes being identified.

	23 – 0 (R)
LCI Buffer Block0 dw0	Energy value for Block 0, code 0, on-time
LCI Buffer Block0 dw1	Energy value for Block 0, code 0, late-time
.....	.....
LCI Buffer Block0 dw14	Energy value for Block 0, code 7, on-time
LCI Buffer Block0 dw15	Energy value for Block 0, code 7, late-time
.....	.....
LCI Buffer Block7 dw0	Energy value for Block 7, code 0, on-time
LCI Buffer Block7 dw1	Energy value for Block 7, code 0, late-time
.....	.....
LCI Buffer Block7 dw14	Energy value for Block 7, code 7, on-time
LCI Buffer Block7 dw15	Energy value for Block 7, code 7, late-time

Figure 42: LCI Buffer Memory Map

5.7.6 PSC Search Buffer

The PSC Search Buffer contains 5120 words, one for every ½ chip in a WCDMA radio time slot. The first location contains the energy value at zero offset (modulo 2560 chips) from GCC, the next at ½ chip, and so on. The PSC Search Buffer is accessible to the host processor only when the PSC Search task is not active. Accessibility is indicated by the PSC Buffer status bit in the CCP Status register; see Section 5.5.1.

	15 – 0 (R)
PSC Search Buffer w0	PSC Search result for offset 0 relative to GCC
PSC Search Buffer w1	PSC Search result for offset ½ relative to GCC
.....	.....
PSC Search Buffer w5118	PSC Search result for offset 2559 relative to GCC
PSC Search Buffer w5119	PSC Search result for offset 2559 ½ relative to GCC

Figure 43: PSC Search Buffer Memory Map

5.7.7 SSC Search Buffer

The SSC Search Buffer contains 1024 32-bit words arranged as eight 4-slot circular buffers, one for each SSC ID. There are 16 symbols per slot of each the PSC and SSC; complex symbols for both the PSC and SSC are stored in the buffer. The first half of the SSC Search Buffer, 512 32-bit words, contains the PSC symbols and the second half the SSC symbols.

First half	32 – 16 (R)	15-0(R)
SSC Circular Buffer 0 w0	SSC Symbol (real part)	SSC Symbol (imaginary part)
SSC Circular Buffer 0 w1	SSC Symbol (real part)	SSC Symbol (imaginary part)
...	...	...
SSC Circular Buffer 0 w15	SSC Symbol (real part)	SSC Symbol (imaginary part)
...		
SSC Circular Buffer 7 w0	SSC Symbol (real part)	SSC Symbol (imaginary part)
SSC Circular Buffer 7 w1	SSC Symbol (real part)	SSC Symbol (imaginary part)
...	...	...
SSC Circular Buffer 7 w15	SSC Symbol (real part)	SSC Symbol (imaginary part)
Second half		
SSC Circular Buffer 0 w0	PSC Symbol (real part)	PSC Symbol (imaginary part)
SSC Circular Buffer 0 w1	PSC Symbol (real part)	PSC Symbol (imaginary part)
...	...	...
SSC Circular Buffer 0 w15	PSC Symbol (real part)	PSC Symbol (imaginary part)
...	...	...
SSC Circular Buffer 7 w0	PSC Symbol (real part)	PSC Symbol (imaginary part)
SSC Circular Buffer 7 w1	PSC Symbol (real part)	PSC Symbol (imaginary part)
...	...	...
SSC Circular Buffer 7 w15	PSC Symbol (real part)	PSC Symbol (imaginary part)

Figure 44: Secondary Search Code Buffer Format

6. Tasks

This section describes how tasks are transferred into the CCP, when they begin executing, and when they complete. It also describes the task set available.

The CCP supports the following tasks:

- Finger, including support for EOL (early-ontime-late) measurement
- Delay Profile Estimation (DPE) ("multipath search")
- Primary Search Code (PSC) Search ("stage 1 search")
- Secondary Search Code Search ("stage 2 search")
- Long Code Identifier (LCI) Search ("stage 3 search")
- Paging Indication Channel (PICH) Despreading

6.1 Updating the Task Buffer and Associated Configuration Memories

The CCP includes a ping/pong Task Buffer that can contain up to MAX_TASKS tasks. Tasks are written to the Task Buffer by software, and after they are loaded into the hardware side, they can be executed by the CCP. The loading is requested by software using the Task Request bits along with a Task Request ID, and occurs at the next Task_Update boundary following the write of the request. This is the basic mechanism for loading tasks into the CCP for execution.

When the Task_Update transfer takes place, the Task_Update_Timestamp register is loaded with the current value of the global chip counter (GCC).

The time interval for the Task_Update boundary is programmable. It must be set at a multiple of the CCP iteration (16 chip periods). This multiple of 16 chips is stored in the double-buffered shadow register Task_Update_Cycle, which is updated by copying the software side of the register to the hardware side on every Task_Update boundary. At reset, the value of Task_Update_Cycle is 4.

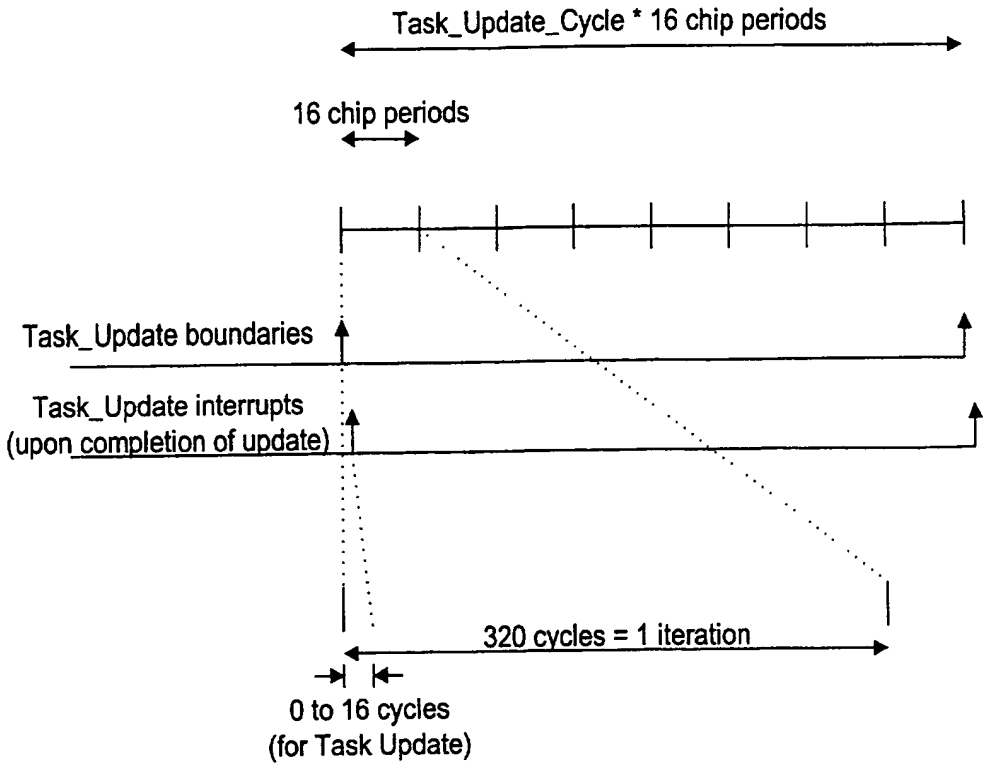


Figure 45: Task Update Timing

The contents of the Task Buffer may be read by the host processor. After the Task_Update interrupt occurs, which signals the completion of the Task_Update actions, reading the Task Buffer will not result in a synchronization problem. See Figure 45.

After power-on reset, the contents of the Task Buffer are undefined, the enable status bits of all tasks in the execution buffer are disabled, and all Task Request bits are cleared. Tasks will begin being executed once the tasks are written to the Task Buffer, Task Requests to load and start are made, and the CCP is started.

6.2 Task Management

Most tasks have an ID associated with them which is used by the CCP to identify which configuration parameters are associated with them and where the results of the tasks are to be placed. The same ID should not be duplicated on two tasks of the same type which run simultaneously. The results of such a situation are not defined.

There are four types of task-level ID's:

- Finger ID
- DPE Search ID
- LCI Search ID
- SSC ID

The Finger ID is used by Finger tasks and by the PICH Despeader task. The Finger ID of a task is used to index coherent scratch memory, internal scratch memory, FSB Configuration Table, FSB Status, Finger Interrupt Table, and EOL Buffer. It is also used to distinguish different Finger task events at the interrupt FIFO's.

The DPE Search ID is used by DPE tasks. Any running DPE tasks must use unique DPE Search ID's. The Search ID of a task is used to index internal scratch memories, and DPE Buffer. It is also used to distinguish different DPE task events at the interrupt FIFO's.

The LCI Search ID is used by LCI tasks. Any running LCI tasks must use unique LCI Search ID's. The Search ID of a task is used to index internal scratch memories, and LCI Buffer. It is also used to distinguish different LCI task events at the interrupt FIFO's.

The SSC ID is used to distinguish different SSC Search task events at the interrupt FIFO's and to index the SSC Buffer.

In each Walsh set, each Walsh sub-task has a unique Walsh ID to distinguish itself from other Walsh sub-tasks. This is discussed in detail in Section 6.3.2.

6.2.1 Starting and Stopping Tasks

The CCP maintains a MAX_TASKS-bit Task Start/Stop Status register which reflects which tasks are "enabled." Enabled tasks are those that are "Running", "Waiting to Run", or "Waiting to Stop". The CCP only fetches a task if it is enabled. Hence, disabled tasks are skipped over and do not expend CCP cycles.

The CCP can autonomously disable/stop tasks which have completed: PSC Search, DPE, LCI and PICH. All other tasks must be stopped by the host. The CCP cannot autonomously start any tasks.

The host processor requests the starting and stopping of CCP tasks at the Task_Update boundary. All tasks may be started or stopped on an individual basis. The host processor

sets the appropriate Task Request Bit and writes the type of request to the Task Request ID register for the task.

A task that is enabled may or may not be actively running; it depends on whether its start time has been reached. If the start time has not been reached, then the enabled task is "Waiting to Run", and one CCP cycle is expended per task. A task which is "Running" or "Waiting to Stop" is actively running and may consume several CCP cycles.

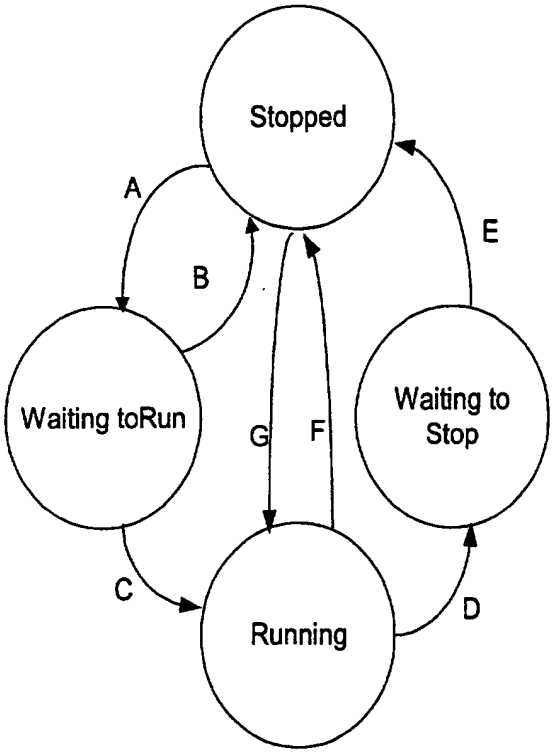


Figure 46: Task Start/Stop State Transition Diagram

Description of task state transitions:

A- from Stopped to Waiting-to-Run: When a synchronous start is requested.

B- From Waiting-to-Run to Stopped: When immediate stop is requested.

C- From Waiting-to-Run to Running: When slot activation time arrives.

D- From Running to Waiting-to-Stop: When synchronous stop is requested.

E- From Waiting-to-Stop to Stopped: When slot activation time arrives or immediate stop is requested.

F- From Running to Stopped: When immediate stop is requested or task completes.

G- From Stopped to Running: When immediate start is requested

6.2.2 Synchronous Task Reloading

A Task may be synchronously reconfigured with reference to its own slot/frame timing. This can be used to support compressed mode as well as other synchronous reconfigurations. A Task can start, stop, or reload configuration on slot boundaries. This is accomplished by requesting the appropriate action for that task and identifying the slot boundary for that action to occur.

If a Walsh table needs synchronous modification, a new Walsh pointer and new Walsh entry should be reloaded. Likewise, if a FSB Configuration Table entry needs modification, the FSBC ping/pong entry for that task, which is not in use, should be modified, and reloaded.

When a Finger goes in and out of compressed mode, it may need to allocate additional memory to accommodate the doubling of number of symbols with the reduction of SF. The unused ping/pong FSB Configuration Table entry may be modified, and the Finger Task reloaded so that the new FSBC is referenced.

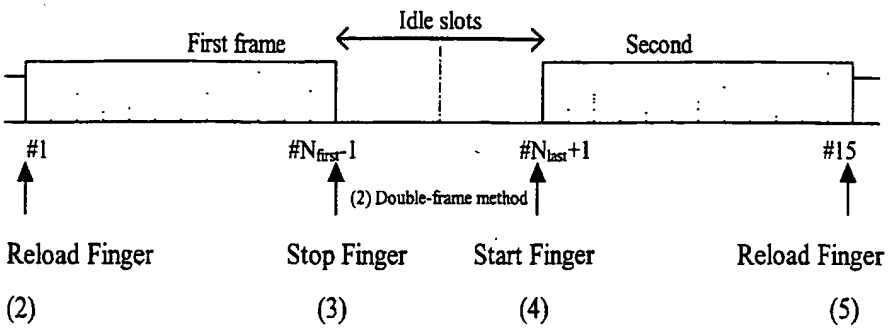


Figure 47: Finger Modifications in Compressed Mode

Summary of Steps to Modify Finger during Compressed Mode:

1) Set Finger nominal conditions, set nominal SF, start Finger.

2) When notified of compressed mode, during last normal frame, write a new task description to the Task Buffer which contains the compressed-mode information to be reloaded as its task description at the end of slot 14 (15th slot), including change of SF, Walsh pointer, and FSBC pointer as necessary. Write a Task Request to synchronously reload the task at the end of slot 14.

3) After that swap has occurred (and its slot ID is no longer needed), write a Task Request to synchronously stop the task at the end of the appropriate slot.

4) After the task stops (and its slot ID is no longer needed), write a Task Request to synchronously start the task (again) at the appropriate slot. (The old task description is ready to run again with no modifications required).

5) After the task starts, write Task Request to synchronously reload the task (back to the nominal conditions) at the end of slot 14 (15th slot). (The original, nominal task description is waiting in the software side of the Task Buffer, ready to run again with no modifications required)

6.2.3 Task Start Time

All CCP tasks may be started immediately, on the Task_Update boundary at which the request is made, or synchronously to its own slot timing. The starting slot may be radio slot 0 through slot 14 or the next radio slot (signified by using slot 15).

A task which is "Waiting to Run" expends one CCP cycle until it enters the "Running" state, at which time it uses the number of CCP cycles determined by its specifications.

In some situations, an immediate start may be desirable – the Finger task for example. For other tasks, however, an immediate start will result in erroneous results.

6.2.4 Task End Times

Some tasks execute without end, others have well defined end times.

- Finger: continuous
- DPE: one-shot
- PSC: one-shot
- SSC: continuous
- LCI: one-shot
- PICH: one-shot

6.3 CCP Task Description

Note: For compatibility with future versions, all "unused" bit fields should be set to zero.

6.3.1 Finger Task

The Finger task is used for demodulating fingers and EOL measurement (DLL support). Despread finger symbols are stored in the Finger Symbol Buffer. EOL measurements are output to the EOL Buffer.

The Finger task can activate up to three interrupts: Pilot, TPC, and End-of-Slot. These are enabled and specified using the Finger Interrupt Table.

EOL results are output once per frame. The finger tasks have no stored history; results are based on the current radio slot number. This means that if a finger is started at other than a frame boundary, it will become re-synchronized on the next frame boundary. After

completing the first full frame the EOL results will be correct. (However, symbol results are valid immediately).

For EOL, the number of time-slots to coherently accumulate the pilot may be N_{TS} = 0.4, 1,2,3,5,7,15. Hence, per frame, there are ***round-down***(15/ N_{TS}) energy accumulations. If N_{TS} does not divide into 15, the last ***remainder***(15/ N_{TS}) slots of pilots in the frame are not used in the energy accumulation.

The slot interrupt of the Finger task is intended to be used to service the FSB, retrieving a slot of symbols which have been completed. The information stored in the interrupt FIFO includes the buffer slot number to assist in identifying the location of the completed symbols. Due to the fact that there are 15 slots per frame, the first slot of data processed after a finger task begins running will not necessarily be in the first slot of the multi-slot circular buffer defined in its FSB Configuration entry. Therefore, the host processor must obtain this information from the interrupt FIFO. After the first slot of information is stored, the subsequent slots are stored as would be expected in a circular buffer arrangement.

The maximum energy value of for each Finger ID/ Walsh ID combination is dumped, once per slot, into a four slot circular buffer called the Finger Max Buffer. This data is to assist in combining RAKE symbols.

	15 - 12	11 - 8	7 - 6	5 - 0
Finger_Task_w0	Opcode	unused	Input ID	Finger ID

	15 - 12	11 - 8	7 - 3	2 - 0
Finger_Task_w1	Spreading factor	N_{TS}	Walsh Pointer (5 LSB's)	Sample #

	15 - 12	11 - 10	9	8	7 - 0
Finger_Task_w2	Unused	Timing Adjust	FSBC Ping/Pong	Compressed	Long Code Offset

	15 - 0
Finger_Task_w3	Frame Offset

	15	14	13	12-10	9 - 0
Finger_Task_w4	Unused	EOL_en	Pilot location	N_{walsh}	Long Code ID

Figure 48: Finger Task Format

The finger task includes the following parameters:

- task opcode – 0100
- Input ID: selects a particular Input Buffer of I/Q data to process.
- Finger ID: A user defined field that is a unique identification number for all Finger tasks.
- Spreading factor: 4, 8, 16, 32, 64, 126, 256, 512, mapped in order to bit values "000" through "111".
- N_{TS}: (pertains to EOL processing only) the number of time-slots to coherently accumulate the pilot, if coherent accumulation chosen for EOL (see Walsh sub-task). This field is limited to the following values:

N_{TS}	Coherent length (time slots)
"0000"	0.4
"0001"	1
"0010"	2
"0011"	3
"010X"	5
"011X"	7
"1XXX"	15

- Walsh Pointer LSB's: selects one of 32 sets of Walsh sub-tasks in the Walsh Table. If TOTAL_WALSH is > 32, then the MSB of the Finger ID is appended to the 5 LSB's in this field to form a Walsh Pointer.
- Sample#: selects which sub-chip samples to process.
- Compressed: A "1" indicates finger is in compressed mode, which affects EOL processing. If the finger is starting, then the coherent and non-coherent accumulations are cleared. While the finger is running, the running non-coherent sum is dumped each time a new computation is completed.

- Timing_Adjust: these bits reflect a change in sampling time for the Finger task. The options are no change, +/- delta, where delta is 1 sub-sample of a chip. The timing update is not reflected in the sample field of the task; the information is stored internally to the CCP. The following values are used:
  - "00": no change
  - "01": +1 chip sample adjustment (to later sample)
  - "1X": -1 chip sample adjustment (to earlier sample)

  To correctly adjust the Finger tasks timing, the Task Timng Adjust Request bit must be set for the task, as well as any changes to the Finger task's sampling time and long-code offset fields of the task description written to the Task Buffer and swapped-in. The Timing Adjust Request informs the CCP of the changes to sampling time and long-code offset of the Finger task that have just happened so that it could modify its internal processing accordingly. If changes to either the sampling time or long-code offset are made, and a Task Timing Adjust Request not made, the Finger task may output erroneous results and trigger interrupts at incorrect times.

- FSBC ping/pong: A '0' indicates to use the ping FSB Configuration buffer to determine where to output symbols, a '1' indicates to use the pong FSB Configuration buffer.

- Long Code Offset: specifies the offset of long code start with respect to the Frame Offset. This is always a multiple of 256 chips, so the field specifies the multiple of 256 chips from the Frame offset. The Long Code start is always earlier than the Frame Offset, so the field is specifing how much earlier (unsigned).

- Frame Offset: specifies the offset with respect to the global chip count (not from the beginning of the long code) at which the frame begins. The four MSB's specify the offset in radio time slots; the remaining LSB's specify the additional sub-slot in chips.

- N_{walsh}: the number of active Walsh sub-tasks minus one. This parameter specifies that the first N_{walsh} + 1 Walsh sub-tasks are all active. For finger tasks with Finger ID's from 32 to 63, this field is limited to "000", "001", "010", and "011".

- EOL_en: This parameter specifies if one of the Walsh sub-tasks is an EOL Walsh sub-task. If "1", the first Walsh sub-task must be the EOL Walsh sub-task. EOL processing is decimated by 4 when the spreading factor is 4, and decimated by 2 when spreading factor is 8 (a limitation which applies only to DPCCH, not CPICH).

- Pilot Location: selects Pilot region 0 or 1 from Pilot-TPC Position Table, for use in the determination of a Pilot interrupt for the Finger task (the EOL processing may use a different Pilot Location, which is specified its the Walsh sub-task).

- Long Code ID: specifies Gold code used for long-code scrambling.

Cycles per CCP iteration:

This task requires a number of cycles equal to the number of non-EOL sub-tasks plus the number of cycles required for an EOL Walsh sub-task (only one allowed per Finger task). An EOL Walsh sub-task takes three cycles.

6.3.2 Walsh Sub-Task

The Walsh sub-task format is used to specify the entries in the Walsh Table. Entries in the Walsh Table are used for Finger and DPE tasks. If there is an EOL Walsh sub-task, it must be at the first location.

If fewer than the maximum number of entries (8 or 4) are needed, there are two options: 1) the Walsh sub-tasks must appear at the beginning of the Walsh set and be consecutive, the N_{walsh} field is set to the appropriate size; or 2) any Walsh sub-task which is not used in the first N_{walsh} must have its Walsh enable bit cleared to indicate that it is disabled. The drawback to the second method is that a cycle is expended for each of the N_{walsh} even those sub-tasks that are disabled.

Walsh sub-tasks are operated in consecutive address order within a set.

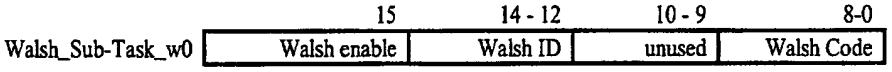


Figure 49: Walsh Sub-Task Format for non-EOL, non-DPE

The Walsh sub-task parameters for Finger tasks (non-EOL):

- Walsh enable. Specifies if this Walsh sub-task is enabled or disabled. "1" means enabled.

- Walsh ID: A user defined field that is unique in a particular Walsh sub-task set of the Walsh Table. It may be reused in other Walsh sub-task sets in the Walsh Table.

- Walsh Code: specifies the Walsh-Hadamard code number.

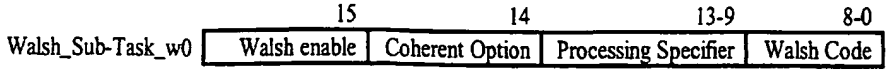


Figure 50: Walsh Sub-Task Format for EOL, DPE

The Walsh sub-task parameters for Finger EOL tasks and DPE tasks:

- Walsh enable. Specifies if this Walsh sub-task is enabled or disabled. "1" means enabled.
- Walsh Code: specifies the Walsh-Hadamard code number.
- Coherent Option: if this bit is set, the coherent processing for more than one symbol is selected (implies a Pilot region), otherwise, coherent processing is limited to one symbol, and energies are accumulated over many symbols (Pilot or Non-Pilot region).
- Processing Specifier: For Multi-Symbol Coherent Processing (Pilot-only):

Figure 51: Processing Specifier for Multi-Symbol Coherent Processing

- Pilot Region selection. Chooses one of two Pilot regions specified in "Pilot-TPC Location Table".
- Pilot Bits selection: Selects one of four options (values "1" through "4") in the Pilot Bits Table, or the Common Pilot Channel (value "0").
- Antenna Diversity Enable: if "1", diversity pilot processing is enabled, energies from two antennas are combined.
- Processing Specifier: For Single-symbol Coherent Processing (Pilot or Non-Pilot):

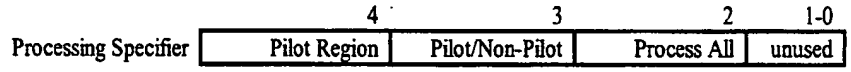


Figure 52: Processing Specifier for Single-Symbol Coherent Processing

- Pilot Region selection. Chooses one of two Pilot regions specified in "Pilot-TPC Location Table".

- Pilot/Non-Pilot: if "1" selects Pilot region as defined in Pilot region selection, if "0" selects the region outside of the Pilot region defined. If the Process All field is "1", this bit is ignored.

- Process All: if "1" all the symbols in the slot are processed, if "0" only the symbols in the region defined by the Pilot Region and the Pilot/Non-Pilot bit are processed.

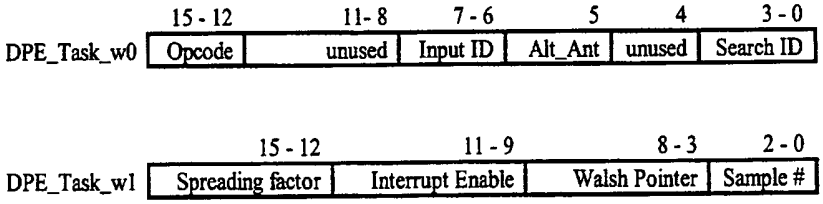
6.3.3 Delay Path Estimate (DPE) Search Task

The DPE Search measures path energies in a specified window of offsets. Two offsets are searched in each CCP cycle. It supports the determination of the strongest paths on any Walsh code channel (which may or may not be beam-formed). When two input antennas are present, the Alternating Antenna Mode can be selected in which alternate symbols are despread from alternate antennas. Results are stored in the DPE Buffer.

Each DPE Search task has a unique DPE Search ID to distinguish itself from other DPE tasks. The Search ID controls where in the DPE Buffer the results are output. The Search ID must not be changed while the DPE task is running – erroneous results will be produced.

The DPE Search Task operates in one-shot mode. When Pilot symbols are accumulated coherently over multiple symbols, (specified by Coherent Option='1' in the Walsh sub-task), N_{TS} controls the number of radio time slots of coherent accumulation, and results are output $N_{TS} * N_{NTS}$ radio time slots after the start of the task. When symbol energies are accumulated, (specified by Coherent Option='0' in the Walsh sub-task), N_{TS} is ignored and a results are output N_{NTS} radio time slots after the start of the task. The values for N_{TS} and N_{NTS} are read from the DPE Energy Accumulation Table referenced by the Search ID.

The DPE Search is decimated by 4 when the spreading factor is 4, and decimated by 2 when spreading factor is 8 (a limitation which applies only to DPCCH, not CPICH).



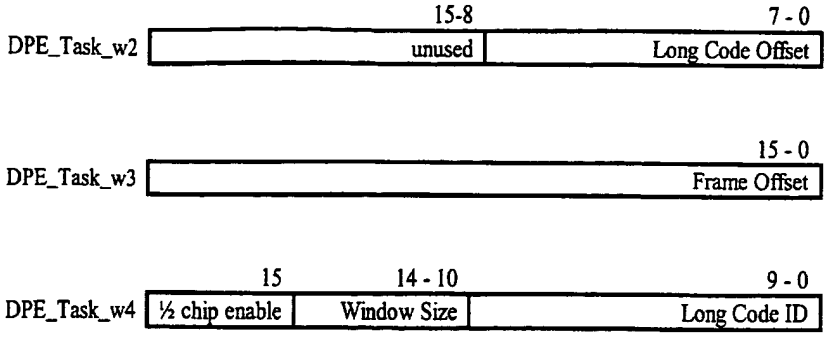


Figure 53: DPE Search Task Format

The DPE Search task consists of several parameters:

- task opcode – 0011
- Input ID: selects which Input Buffer.
- Alt_Ant: a "1" selects Alternating Antenna Mode from two Input Buffers: 00 and 01
- DPE Search ID: distinguishes different DPE tasks.
- Spreading Factor: 4, 8, 16, 32, 64, 126, 256, 512, mapped in order to bit values "000" through "111".
- Interrupt Enable: characterizes interrupt. The MSB enables interrupt. The two LSB's specify which interrupt FIFO.
- Walsh Pointer: selects Walsh sub-task set. The Walsh sub-task in the first location is used, others are ignored, Walsh enable field of sub-task is ignored.
- Sample #: selects which sub-chip sample to be "on-time".
- Long Code Offset: specifies the offset of long code start with respect to the Frame Offset. This is always a multiple of 256 chips, so the field specifies the multiple of 256 chips from the Frame offset. The Long Code start is always earlier than the Frame Offset, so the field is specifing how much earlier (unsigned).
- Frame Offset: specifies the offset with respect to the global chip count (not from the beginning of the long code) at which the frame begins. The four MSB's specify the offset in radio time slots; the remaining LSB's specify the additional sub-slot in chips.
- ½ chip Enable: enables the processing of samples at ½-chip resolution.

- window size – (5 bits), 0<=n<31, gives a window size of 16*(n+1) relative to Long Code Offset.
- Long Code ID: specifies Gold code used for long-code scrambling.

Cycles per CCP iteration:

This task requires a number of cycles equal to half the number of offsets in the window size, regardless of whether ½-chip processing is enabled.

6.3.4 Primary search Code (PSC) Search Task

The PSC Search task supports the locating of PSC in the WCDMA standard, which determines slot timing. The results are stored in the PSC Search Buffer and can be read by the processor once the task completes. At any time, only one running PSC Search Task is allowed and it must be the first task to run in order for the post-processing hardware to have time to complete its processing.

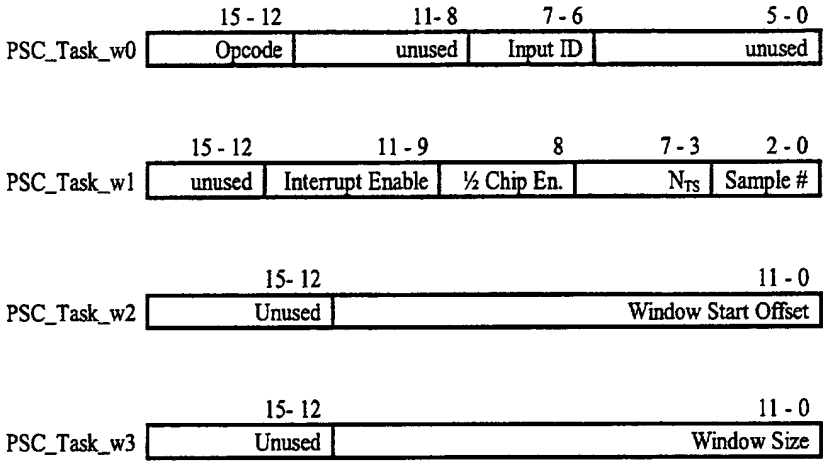


Figure 54: PSC Search Task Format

The parameters for PSC Search are:

- task opcode: 0001
- Input ID: selects which Input Buffer.

- Interrupt enable: characterizes interrupt. The MSB enables interrupt. The two LSB's specify which interrupt FIFO.
- ½ chip enable: enables the processing of samples at ½-chip resolution.
- N_{TS}: number of radio time slots to accumulate energy.
- Sample #: selects which sub-chip sample to be "on-time".
- window start offset: specifies the start time for the task in chips.
- window size: specifies the number of chips to be included in the window.

Cycles per CCP iteration:

The PSC Search task requires 16 cycles.

Activation Time:

This task is activated when the GCC modulo 2560 is equal to the value specified in the window start offset field.

6.3.5 Secondary Search Code (SSC) Search Task

The SSC Search task performs part of the stage-two WCDMA base-station search. It operates only on the masked-out symbol of a Perch channel and outputs 16 symbolsde-spread by the SSC (of 16-chip length), as well as 16 symbols de-spread by the PSC, per radio time slot. In this way, the SSC Search task operates similarly to a Finger task. The remaining stage-two processing – applying the second level of SSC coding, the Walsh-Hadamard transform, and matched-filtering with Comma-free code – takes place in the host processor or in hardware outside of the CCP.

The masked-out symbol location is specified in the Search Code Symbol Location register.

The SSC Search task uses a SSC ID that must be unique to any other SSC Search tasks. For data processing, the SSC Search task does not use a Walsh Pointer. The resulting SSC and PSC are placed in a four slot circular buffer within the SSC Buffer.

The SSC Search activates an end-of-slot interrupt when a complete slot of data is ready in the SSC Buffer.

The SSC Search task starts at the identified slot boundary if the "new" bit is set. It can also be started at any task update boundary. If the new bit is not set and the Stop Enable field of the task is set, then the task will stop on the indicated slot boundary.

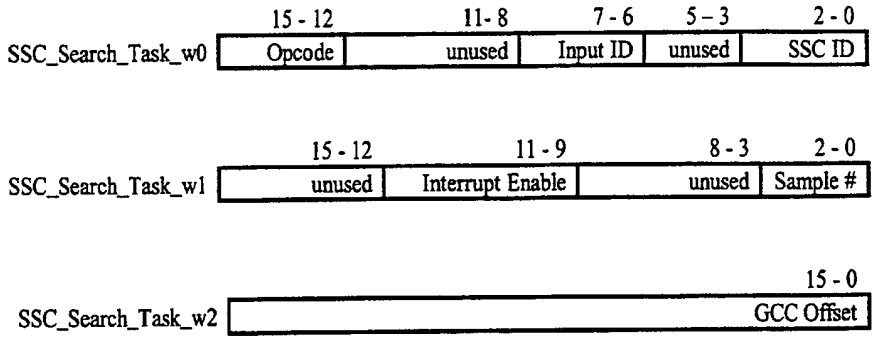


Figure 55. SSC Search Task Format

Parameters:

- Opcode: 0101
- Input ID: selects which Input Buffer.
- SSC ID: A user defined field that is a unique identification number for all SSC tasks.
- Interrupt Enable: The MSB enables the end-of-slot interrupt. The two LSB's specify which interrupt FIFO.
- Sample #: selects which sub-chip sample to be "on-time".
- GCC Offset: This parameter causes the "frame" timing to be like a finger with LC Offset = GCC Offset and Frame Offset = 0. This parameter in effect establishes "frame" timing. Of course, frame timing is arbitrary for SSC Search, since the purpose of SSC search is to establish frame timing. But having a specific "frame" time allows the SSC Buffer and SW to have some time reference in relation to the new frame timing that will be established.

  The four MSB's specify the offset in radio time slots; the remaining LSB's specify the additional sub-slot offset in chips.

6.3.6 Long Code Identifier (LCI) Search Task

The LCI Search task supports the determination of the long code from a group of long codes ("stage 3 search") which were determined in "stage 2 search" by processing the CPICH. The results are stored in the LCI Buffer.

Each LCI Search task has a unique LCI Search ID to distinguish itself from other LCI tasks. The Search ID controls where in LCI Buffer the results are output. The Search ID must not be changed while the LCI task is running – erroneous results will be produced.

The LCI Search Task operates in one-shot mode. Pilot symbols are accumulated coherently over multiple symbols, N_{TS} controls the number of radio time slots of coherent accumulation, and results are output $N_{TS} * N_{NTS}$ radio time slots after the start of the task. The values for N_{TS} and N_{NTS} are read from the LCI Energy Accumulation Table referenced by the Search ID.

The LCI Search task always processes the Common Pilot Channel. Antenna diversity is an option.

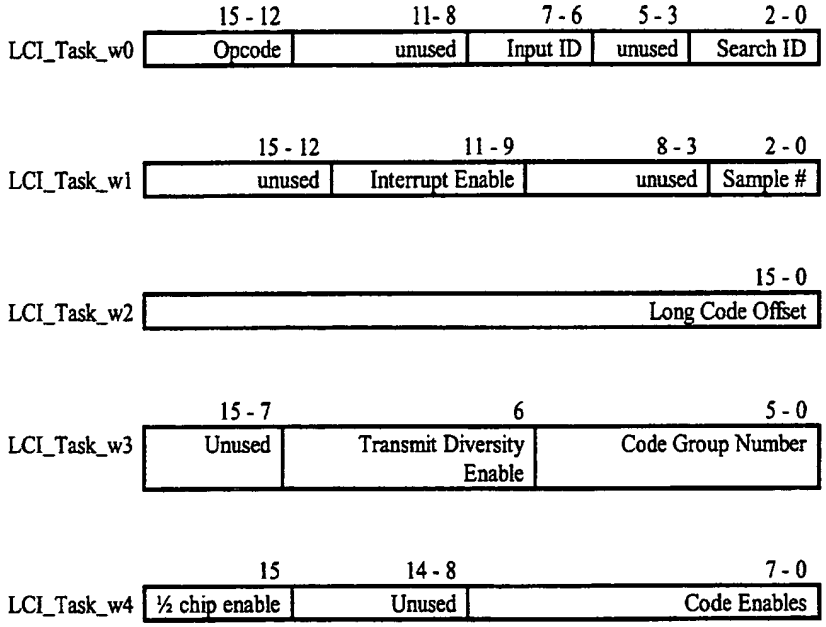


Figure 56: LCI Search Task Format

The parameters for LCI Search are:

- task opcode - 0010
- Input ID: selects which Input Buffer.
- LCI Search ID: distinguishes different LCI tasks.
- Interrupt Enable: characterizes interrupt. The MSB enables interrupt. The two LSB's specify which interrupt FIFO.
- Sample #: selects which sub-chip sample to be "on-time".
- Long Code Offset: specifies the offset of long code start with respect to the global chip count. The four MSB's specify the offset in radio time slots; the remaining LSB's specify the additional sub-slot offset in chips.
- Transmit Diversity Enable: if "1", diversity pilot processing is enabled, energies from two antennas are combined.
- Code Group Number: specifies which group of 64 code groups, having 8 long codes each, to process and test.
- ½ chip enable: if "1", enables processing of ½-chip late samples.
- Code Enables: specifies which of the 8 long codes to test, a single bit enables the testing of each long code, bit 7 for code 7... downto bit 0 for code 0; a '1' indicates the code should be tested.

Cycles per CCP iteration:

This task requires a number of cycles equal to the number of long codes to complete.

6.3.7 Paging Indication Channel (PICH) Despreading Task

The PICH Despreading task despreads symbols in a specified window of offsets at ½ chip resolution. Results are stored in the Finger Symbol Buffer according to the Finger ID used, and the Walsn ID pointed to by the Walsh Pointer.

Each PICH task has a unique Finger ID to distinguish itself from other PICH and Finger tasks.

The PICH Task operates in one-shot mode for 1-8 symbols.

15 - 12	11 - 8	7 – 6	5 - 4	3 - 0

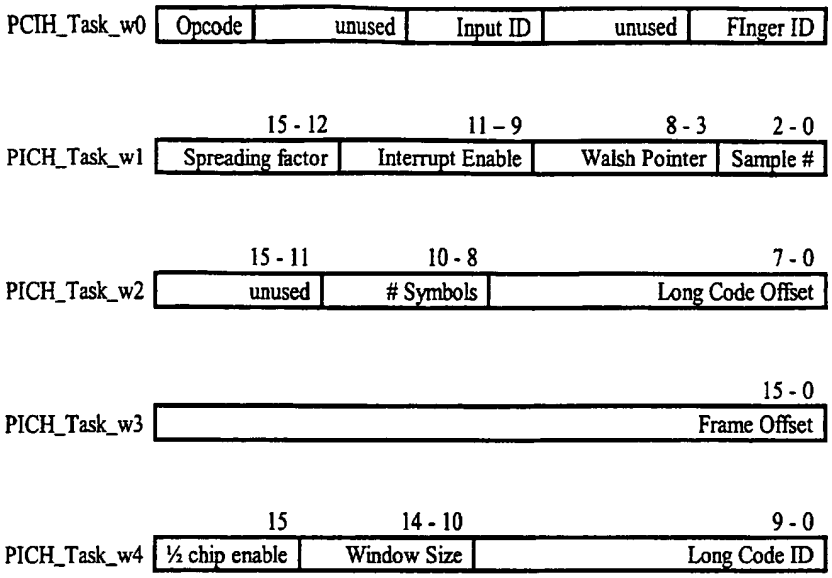


Figure 57: DPE Search Task Format

The PICH task consists of several parameters:

- task opcode – 0111
- Input ID: selects which Input Buffer.
- Finger ID: distinguishes different Finger and PICH tasks.
- Spreading Factor: 4, 8, 16, 32, 64, 126, 256, 512, mapped in order to bit values "000" through "111".
- Interrupt Enable: characterizes interrupt. The MSB enables interrupt. The two LSB's specify which interrupt FIFO.
- Walsh Pointer: selects Walsh sub-task set. The Walsh sub-task in the first location is used, others are ignored, Walsh enable field of sub-task is ignored.
- Sample #: selects which sub-chip sample to be "on-time".
- # Symbols: determines the number of symbols for the instruction to run, "000" thru "111" indicate 1 thru 8.
- Long Code Offset: specifies the offset of long code start with respect to the Frame Offset. This is always a multiple of 256 chips, so the field specifies the multiple of 256

chips from the Frame offset. The Long Code start is always earlier than the Frame Offset, so the field is specifing how much earlier (unsigned).

- Frame Offset: specifies the offset with respect to the global chip count (not from the beginning of the long code) at which the frame begins. The four MSB's specify the offset in radio time slots; the remaining LSB's specify the additional sub-slot in chips.
- ½ chip Enable: enables the processing of samples at ½-chip resolution.
- window size – (5 bits), 0<=n<31, gives a window size of 16*(n+1) relative to Long Code Offset.
- Long Code ID: specifies Gold code used for long-code scrambling.

Cycles per CCP iteration:

This task requires a number of cycles equal to the number of offsets in the window size, regardless of whether ½-chip processing is enabled.

7. Finger Symbol Buffer

7.1 Overview

The Finger Symbol Buffer stores complex I and Q "symbols" that result from Finger tasks. All symbols – Pilot, TPC, data, etc. – are stored here after they are received and processed by the CCP datapath.

The Finger Symbol Buffer is implemented as many multi-slot circular buffers. There are is total of 20K complex symbols (32 bits/symbol) which can be stored at spreading factors from 8 to 512, or 40K complex symbols (16 bits/symbol) which can be stored at spreading factor 4. In order to minimize the reconfiguration of symbol buffering when fingers or Walsh channels are added and removed, each Walsh channel of each finger uses a different circular buffer to store symbols.

The total size of the FSB is 20K x 32 bit words.

Each Finger ID/ Walsh ID combination specifies a multi-slot circular buffer in the FSB which will be used to store its data. A starting address for the first slot and an offset from the starting address of one slot to the next slot. This allows Walsh channels for the same Finger task to be placed adjacent to one another on a slot by slot basis.

The host processor is responsible to ensure that there is no overlap between buffers for a single Finger ID/ Walsh ID combination and between different Finger ID/ Walsh ID combinations.

Each circular buffer is four radio slots long. The starting slot for data when an instruction is started is determined by hardware, and must be read by software to maintain synchronization. This is due to the combination of a four slot circular buffer and there being 15 radio slots per frame. A super-frame count from 0 to 3 is kept as an adjunct to GCC to provide a reference to the hardware in determining which FSB slot in the circular buffer to place data in.

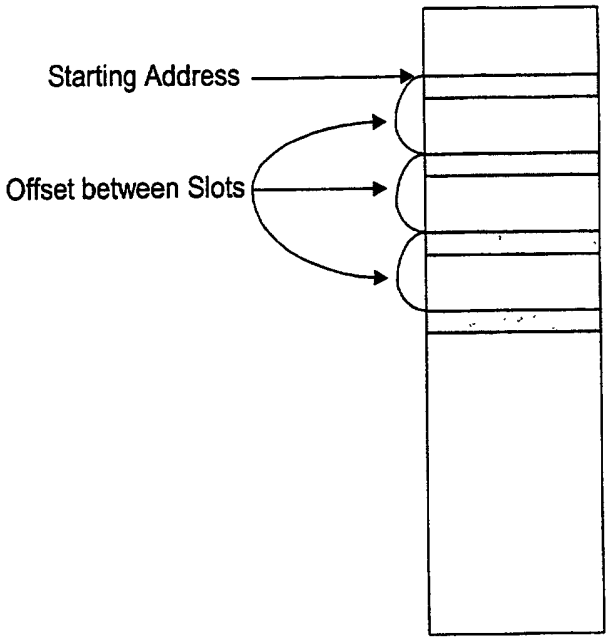


Figure 58. Circular Buffer within Finger Symbol Buffer.

7.2 Configuration of Finger Symbol Buffer

The FSB Configuration Table is used to assign a particular Finger ID and Walsh ID to a particular circular buffer. The FSB Configuration Table is updated at Task-Update time using the Temporary Buffer.

The memory-map of the FSB Configuration Table, is shown in Section 5.2.4.

7.3 Retrieving and Transferring Data from FSB

The Finger Symbol Buffer serves as intermediate storage for downstream symbol processing. In order that symbol processing take place in hardware or software, the FSB is accessible on both the Rhea Bus and the FSB External Bus. The Rhea Bus access allows access by the host processor either directly or via DMA. The FSB External Bus access allows downstream hardware to directly access finger data.

8. Interrupts

There are six interrupt lines from the CCP, which go to an interrupt controller. There are four interrupts which are general-purpose interrupts and come from four FIFOs, as well as a System interrupt and an Error interrupt.

The timing of external interrupts and interrupt events is described in Section 8.

8.1 Interrupt FIFO

Each of four Interrupt FIFOs contains interrupt-event information that comes from CCP tasks. For example, a Finger task can trigger multiple interrupt events that may be mapped to a particular Interrupt FIFO. Each FIFO may store up to $N_{intFIFO} = 16$ task-based interrupt events. Reading the FIFO by the host processor decrements the number of contained events. A FIFO is used because there may be many interrupt events triggered by CCP tasks, and it allows the host processor to optimize the servicing of CCP interrupts.

8.1.1 Mapping task events to FIFO

Each task may map its interrupt events to any interrupt FIFO, using the mechanisms below:

1. Finger task. Finger interrupt events are enabled and mapped using the Finger Interrupt Table. Refer to Section 5.2.5. Finger interrupt events that may be mapped to interrupt FIFO's are

   - Pilot Interrupt event
   - TPC Interrupt event
   - Slot Interrupt event

   These interrupt events may be mapped to any of the FIFO's. For example, mapping them to the same FIFO is allowed.

2. Other tasks. All other tasks can trigger one task-based interrupt event each. Each task's interrupt event is enabled and mapped using the Interrupt Enable field in the task specification.

8.1.2 FIFO Contents

When a task event occurs, the following data is written into the FIFO:

- Task opcode.
- Task ID.
- Interrupt events. For Finger task only; used to specify which of the Finger-task based events. When multiple Finger interrupt events occur in the same CCP cycle, only one FIFO entry is logged. Hence, this field indicates not only which *one* of the events but which *combination* of events. The timing of interrupt events and external interrupts is discussed in Section 8.4.
- GCC. The Global Chip Count at a resolution of 16 chips.

The data above make up one FIFO entry, which is accessed by the host processor on the Rhea Bus using two read accesses. The exact format of these fields is shown in Section 5.6.3.

8.1.3 FIFO Timing

As task events occur, their status data (as described in Section 8.4) are immediately entered into the FIFO that is mapped. However, these entries cannot be read until the next 16-chip boundary of the GCC. Likewise, they activate the FIFO's non-empty status at the next 16-chip boundary. In one CCP iteration, many entries may be written to a FIFO, but they cannot be read out until the next 16-chip boundary.

8.2 System events

System events are:

- Task-Update Interrupt event

8.3 Error events

Error events are:

- Number of cycles exceeded
- Temporary Buffer access while Task Update pending
- FIFO overflow (4 bits)

- Additional error conditions are TBD

8.4 Interrupt Timing

The external interrupts are activated immediately when the interrupt events mapped to them are activated. Activation timing for interrupt events are as follows:

- FIFO (non-empty) event. The non-empty status of a FIFO is only activated at the next 16-chip boundary following the CCP cycle where the task-based event occurs. Note that the task event information is entered into the FIFO when the task-based event occurs, but this new (yet not indicated) entry cannot be read until the next 16-chip boundary (nor does the new entry affect the FIFO's non-empty status).

- System interrupt event. The Task-Update interrupt event occurs at the Task-Update boundary or when the transfer of data from the Temporary Buffer is completed. The Task-Update boundary always resides on a 16-chip boundary.

- Error interrupt event. Activated immediately when a defined system error occurs.

8.5 Clearing events and external interrupts

An external interrupt is cleared when all of the interrupt events that are mapped to it are cleared. Clearing interrupt events is accomplished as follows:

- FIFO (non-empty) event: when all FIFO entries are read out by host processor.

- System interrupt event: when the System Interrupt Event Status register is read by the host processor.

- Error interrupt event: when the Error Interrupt Event Status register is read by the host processor.

9. Glossary

3G	3rd Generation
ABB	Analog Baseband (cf AFE)
A/D (ADC)	Analog to Digital Converter
AFC	Automatic Frequency Control
AFE	Analog Front End (cf ABB)
AGC	Automatic Gain Control
ARIB	Association of Radio Industries and Businesses (Japan)
ARM	Advanced Risc Machine
BS	Base Station
CCP	Correlator Co-Processor
CDMA	Code Division Multiple Access
CPICH	Common Pilot Channel
D/A (DAC)	Digital to Analog Convertor
DBB	Digital Baseband
DLL	Delay Lock Loop
DMA	Direct Memory Access
DPCCH	Dedicated Physical Control Channel
DPE	Delay Profile Estimation
DPP	Delay Path Power
DPPE	see DPE

DSP	Digital Signal Processor/Processing
DSPRDC	DSP Research and Development Center
ETSI	European Telecommunications Standards Institute
FSC	First Short Code
GPS	Global Positioning System
HW	Hardware
ITU	International Telecommunications Union
L1	Layer 1 (physical layer)
L2	Layer 2 (link layer)
L3	Layer 3 (network layer)
LC	Long Code
LFSR	Linear Feedback Shifter Register
MS	Mobile Station
PN	Pseudo Noise
RF	Radio
RTT	Radio Transmission Technology
RX	Receive/Receiver
SC	Short Code
SIR	Signal to Interference Ratio
SW	Software
TI	Texas Instruments

TIA	Telecommunications Industry Association
TRDC	Tsukuba Research and Development Center
TX	Transmit/Transmitter
UMTS	Universal Mobile Telephone System
WBU	Wireless Business Unit (TI)
WCS	Wireless Communications Systems (TI)
WCDMA	Wideband CDMA

10. References

1. S. Sriram. Correlator Co-Processor. Texas Instruments technical activities report. 1998.

2. TIA, TR45.5. The cdma2000 RTT Candidate Submission.

3. placeholder

4. IMT-2000: W-CDMA Specification (specifically DS mode).

5. CCP Output Controller Specifications. Texas Instruments internal document. In work.

6. OMAP Specifications. Texas Instruments internal document.